Escola Politècnica Superior
d'Edificació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

GRAU EN ENGINYERIA EN GEOINFORMACIÓ I GEOMÀTICA
TREBALL DE FI DE GRAU

IMPLEMENTACIÓ D'UN PROGRAMA BASAT EN ELS MÈTODES STRUCTURE FROM MOTION

Projectista/es: David Fernández Bigordà
Director/s: Albert Prades Valls
Convocatòria: Febrer 2020

0 RESUM

En el present projecte es detallen els procediments seguits per a l'obtenció d'un núvol de punts tridimensional dens a partir d'un bloc d'imatges, tot a partir de la implementació de llibreries open source.

L'aplicació desenvolupada és capaç d'obrir un bloc d'imatges, generar els seus punts d'interès, donar-les una orientació, relacionar les diferents imatges entre sí per a extreure coordenades 2d i finalment generar unes coordenades 3d.

La llibreria open source que s'ha fet servir per aquest projecte és l'OpenCV versió 4, una llibreria de programació de codi obert dirigida principalment a la visió per computador en temps real. A part de les llibreries bàsiques que ja venen per defecte en el moment de la instal·lació, també s'ha fet servir una llibreria que s'ha de compilar paral·lelament anomenada SFM (*Structure From Motion*), una llibreria on es troben totes aquelles tècniques fotogrametries que permeten estimar estructures tridimensionals a partir de seqüències d'imatges bidimensionals.

En la primera part de la memòria es fa un estudi de les diferents operacions que s'han de tenir presents per aconseguir el nostre objectiu, i de les diferents maneres que es pot aconseguir. A part de la teoria mencionada al llarg de la memòria, també s'han realitzat diferents experiments paral·lels a l'objectiu final del treball. Aquests experiments s'han realitzat per a fer una reafirmació de la teoria estudiada o per a fer comparacions entre els diferents mètodes d'actuació que estan explicats en la memòria.

En la segona part, s'ha fet servir el programa que es desenvolupa en aquest projecte per a la realització d'un model digital ja conegut, per a després poder fer una comparació i veure si el programa implementat dóna els resultats esperats.

ÍNDEX

1 INTRODUCCIÓ

L'avanç en noves tecnologies, siguin de hardware o de software, amb relació al camp de la fotogrametria o de la visió computacional ha sigut un creixement exponencial des de fa 20 anys. Això a fet que la demanda en productes relacionats a aquest món hagi augmentat, fent aparèixer molts productes nous. Un d'aquests productes són el programes de creació de models numèrics tridimensionals.

Ja sigui per crear models de terreny o per fer estudis d'objectes concrets, els models digitals són una forma d'obtenció d'informació sobre aquella matèria d'estudi. El problema és que qualsevol programa informàtic comercial, té un preu molt elevat. Això pot plantejar un gran problema; i una solució seria la creació d'un programa propi que fes tot allò que es podria fer amb un programa comercial.

És a partir d'aquesta tessitura que apareix la idea de la creació d'un programa per a l'obtenció de models digitals a partir de les tècniques basades en el que en anglès en diuen *Structure from Motion* (SFM). Aquestes eines les trobem en la llibreria de programació lliure OpenCV; unes llibreries que estan orientades a la visió per computador i àmpliament utilitzades en robòtica i visió industrial, per exemple. Ara, amb aquest treball volem acostar aquestes tècniques a l'àmbit de la fotogrametria i la geomàtica.

En aquesta memòria podem trobar tota la teoria necessària per aconseguir el nostre objectiu, l'obtenció d'un model digital a partir d'un bloc fotogramètric, i els resultats que s'han obtingut posteriorment quan s'ha integrat aquesta teoria en un programa informàtic.

2. NUCLI DE LA MEMÒRIA

2.1. Què és OpenCV?

OpenCV (Open Source Computer Vision) és una llibreria de programació de codi obert creada per Intel. Aquesta llibreria consta de diferents algoritmes que permeten la visió per ordinador a temps real, com per exemple: detecció i reconeixement de cares humanes, identificació d'objectes, seguiment de moviments d'objectes, trobar imatges similars en una base de dades d'imatges, etc.

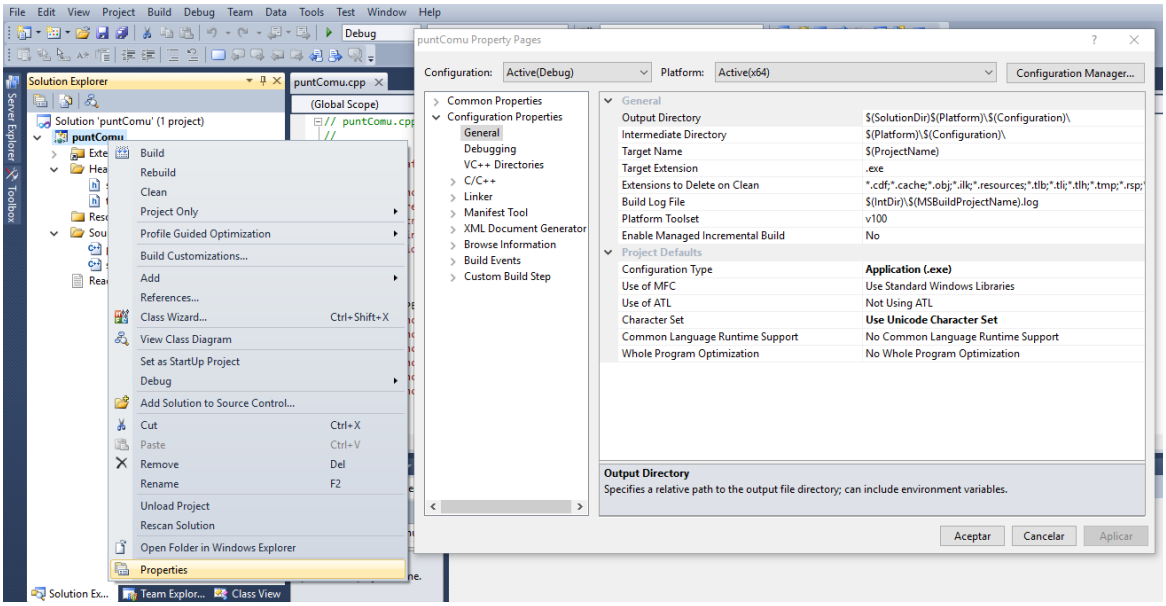
OpenCV és totalment gratuït i es pot fer servir tant com en fins acadèmics com a fins comercials i s'utilitza sota la llicencia BSD (Berkeley Software Distribution). La llibreria OpenCV és multiplataforma, és a dir, suporta les plataformes de Windows, Linux, Mac OS, iOS i Android. En l'àmbit de llenguatge de programació podem utilitzar OpenCV en diversos llenguatges, com ara tal: C#, C, C++, Python o Java.

2.2. Instal·lació de OpenCV

Per a poder treballar amb les llibreries de OpenCV primerament necessitem un entorn integrat de desenvolupament (IDE) on escriurem tot el nostre codi. En el nostre cas utilitzarem el de Windows, Visual Studio, en la versió de 2010.

Seguidament, descarreguem OpenCV en la pàgina web oficial (www.opencv.org), on descarregarem un executable que ens realitzarà una instal·lació dels diferents mòduls, en una carpeta de destí. La versió que utilitzem perquè el programa compili de manera adequada és la versió 3.3.1. Les versions més noves de OpenCV separen els algoritmes per a trobar els punts comuns (SIFT, SURF, ORB) en uns paquets externs que s'han de descarregar a part e incloure en la carpeta on tenim instal·lat el OpenCV; per aquest motiu resulta molt més senzill utilitzar versions antigues perquè tots els algoritmes que necessitem per aquest cas ja vénen dintre de la instal·lació estàndard.

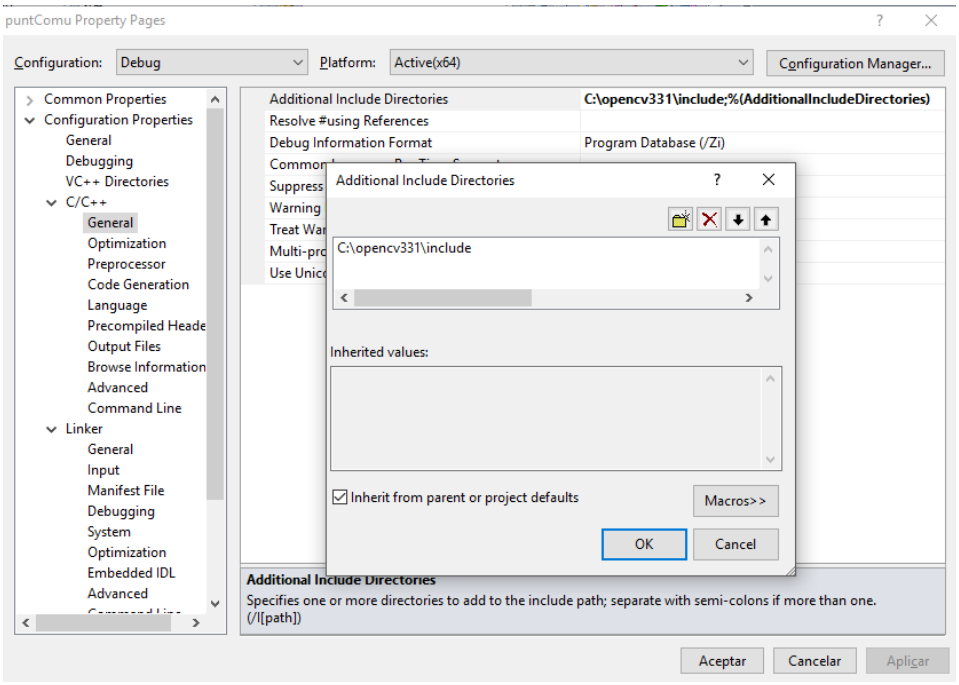
Un cop iniciem el Visual Studio 2010 hem de realitzar un conjunt de passos perquè l'OpenCV funcioni de manera adient. En l'apartat del *Solution Explorer* donarem amb el botó dret, per a desplegar el menú contextual i seleccionar l'apartat de Propietats [Imatge 1].



Imatge 2.1: Propietats del projecte

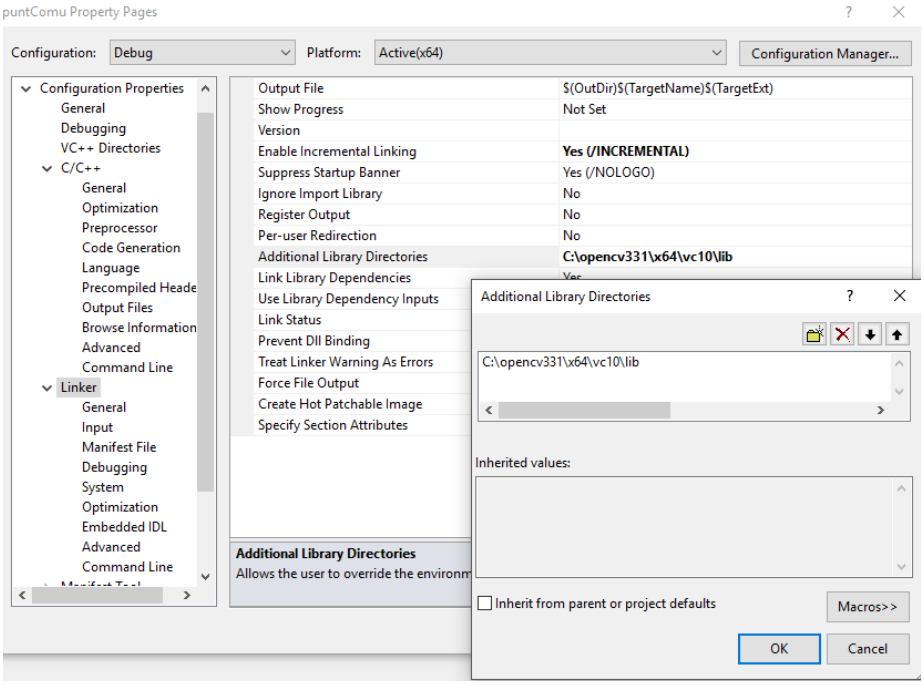
Una vegada dintre de les propietats ens hem de fixar en el desplegable de configuració, ja que tenim el Debug i el Release. Primerament podem realitzar els passos a seguir en el Debug.

Seleccionem en el menú desplegable de configuració la configuració per a Debug, i posteriorment ens fixarem en el menú que trobem en l'apartat esquerre de la finestra. En aquest apartat haurem de realitzar canvis en el C/C++→General i també en Linker→General i Linker→Input. Al accedir a C/C++→General, veurem un apartat que posa *Additional Include Directories* i aquí cliquem sobre el menú desplegable de la casella de la dreta per a seleccionar <Edit>. En la següent finestra emergent que ens apareixerà, copiarem la direcció a la carpeta *include* que trobem dintre de la carpeta OpenCV que hem instal·lat anteriorment [Imatge 2.2].



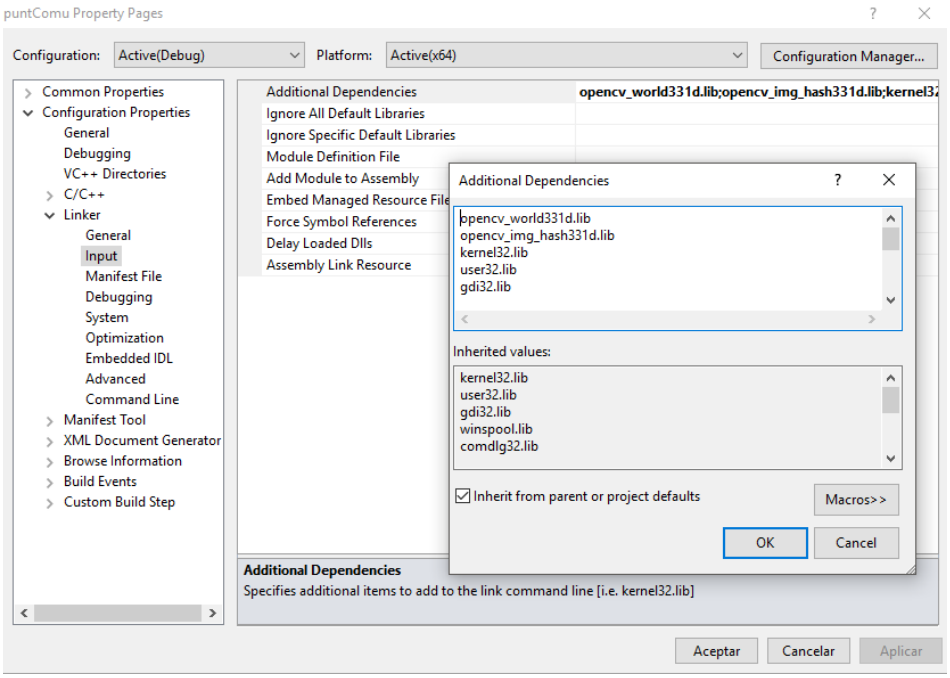
Imatge 2.2: C/C++ -> General

Posteriorment entrem en l'apartat de Linker→General i seleccionem *Additional Library Directories* i despleguem el menú desplegable per a seleccionar l'opció <Edit>. A l'obrir-se la nova finestra hem d'escriure la direcció de la carpeta on es troben les llibreries del OpencCV [Imatge 2.3].



Imatge 2.3: Linker -> General

Finalment entrem en l'apartat de Linker→Input. En aquest apartat fem clic en *Additional Dependencies* i incloem les llibreries. Aquestes llibreries les trobem guardades en la carpeta del OpenCV. Anem a la direcció on tinguem la carpeta i busquem en la carpeta de x64 una segona carpeta que es digui lib. En aquesta carpeta trobem diferents .lib, que són les llibreries que contenen les diferents funcions que ens ajuden a executar el nostre programa. En fixar-nos en tots els .lib podem observar que hi han uns que tenen una d final i uns altres que no; aquesta d final ens indica que aquesta llibreria és per a Debug o per a Release. En el nostre cas copiem el nom de les dues llibreries que ens interessen que són: opencv_world331d.lib i opencv_img_hash331d.lib [Imatge 2.4].



Imatge 2.4: Linker -> Input

A continuació realitzem tots els passos anteriors, però canviant de Debug a Release. Ens haurem de fixar que en el moment que ens trobem en l'apartat d'incloure les llibreries, haurem de copiar el nom de les llibreries que no tinguin la d extra.

Finalment entrem en les carpetes que ens ha generat el nostre propi projecte de Visual Studio i enganxem els .lib en la carpeta de Debug i Release respectivament.

2.3. Què és SFM?

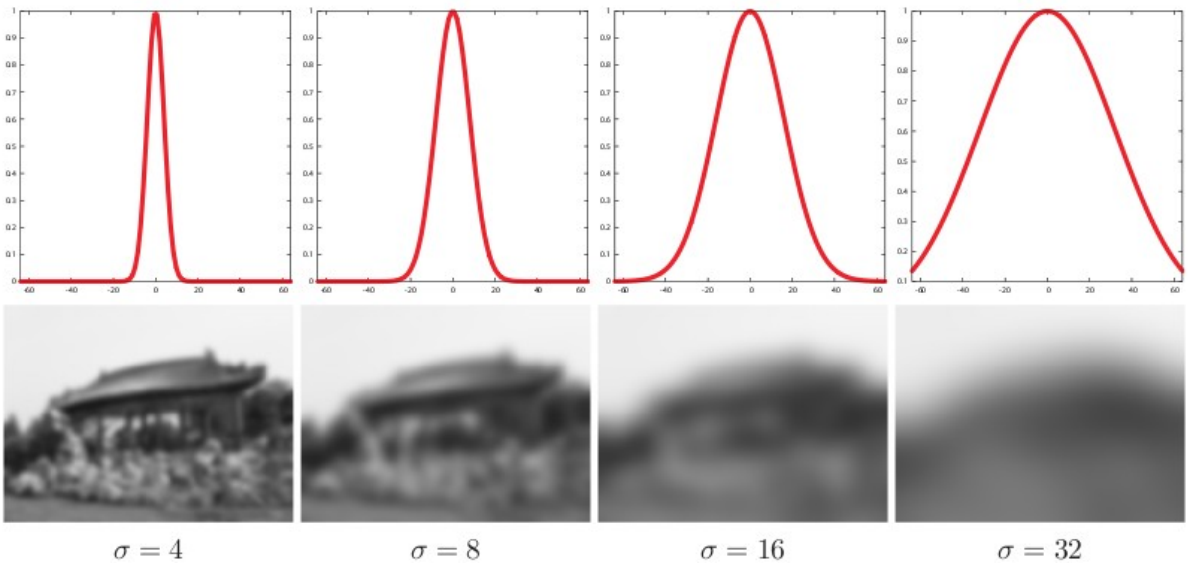
SFM o *Structure From Motion* és una tècnica de fotogrametria de la visió computacional, que ens permet l'obtenció de dades 3D a partir d'una seqüència de dades 2D.

2.4. Generació de KeyPoints

Per a la generació de KeyPoints o punts d'interès, coneixem diversos algoritmes però en aquest cas ens fixarem 3 algoritmes diferents: SIFT, SURF i ORB. Aquests algoritmes són un conjunt d'algoritmes que detecten zones de les nostres imatges que es poden considerar únics o d'interès per un seguit de característiques que només podem trobar en aquell punt concret de les imatges.

2.4.1. Algoritme SIFT

SIFT (*Scale Invariant Feature Transform*) (Alegre & Fernánez-Robles n.d.; Flores & Braun 2011b; Duarte Villaseñor & Chang Fernández 2010; Lindeberg 2012) és un algorisme de visió artificial que serveix per extraure punts d'interès de les imatges. Aquest algorisme va ser publicat originalment per David Lowe en 1999 i, posteriorment, va ser patentat l'any 2004 als Estats Units.



Imatge 2.5: Exemple de funcionament del filtre Gaussià

La primera operació que realitza l'algoritme és la creació de noves imatges a partir de la imatge original. Aquestes imatges derivades es troben a partir de l'aplicació d'un filtre Gaussià [Imatge 2.5], que fa que les imatges resultants es mostrin més suavitzades. Aquest filtre Gaussià ens ajuda a ressaltar les bores dels objectes que tenim en la nostra imatge. Posteriorment d'haver obtingut les imatges filtrades realitzarem la diferència de Gaussià (DoF), a partir dels valors originals de la nostra imatge I (x, y) amb el filtre Gaussià G (x, y, σ):

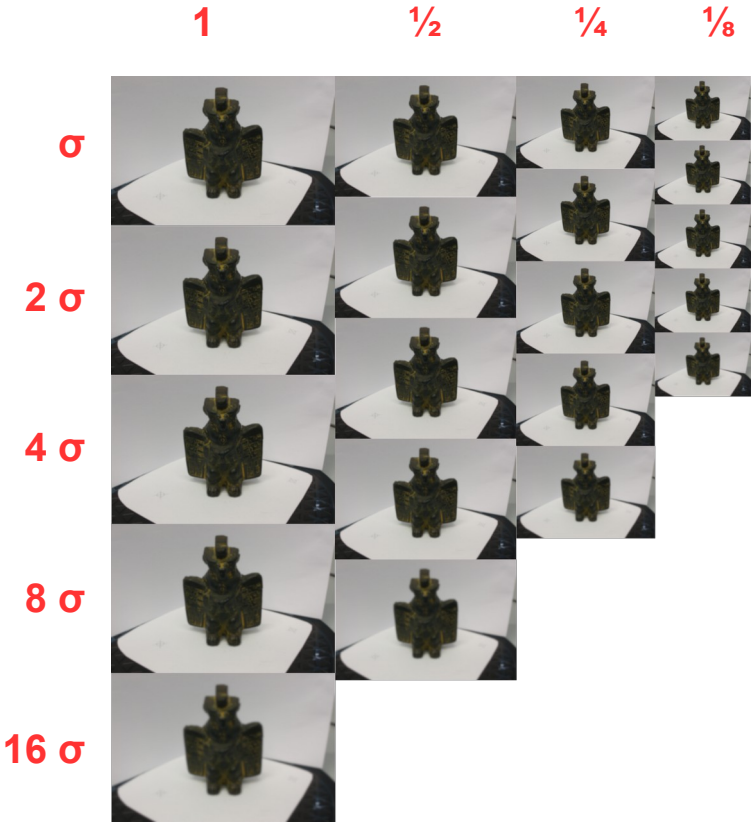
$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}}$$

$$L(x,y,\sigma) = G(x,y,\sigma) \cdot I(x,y)$$

$$D(x,y,\sigma) = L(x,y,k\sigma) \cdot L(x,y,\sigma)$$

On el valor k és un factor constant que multiplica a la desviació típica.

Posteriorment, l'algoritme SIFT redueix la mida de la imatge original a la meitat i torna a aplicar el filtre Gaussià. Aquest procediment es realitzarà un total de quatre vegades organitzant d'aquesta manera les imatges en octaves [Imatge 2.6]. La comparació de totes aquestes imatges resultats en donarà, posteriorment, quins són aquells píxels que podem considerar punts d'interès.



Imatge 2.6: Mostra de les octaves de la imatge de l'experiment

Gràcies ha aquest procediment obtenim un llistat de punts d'interès que poden o no, ser útils com a tal. Els punts d'interès són aquells que tenen un valor més gran en comparació als seus veïns més propers.

Imatge 2.7: Mostra de valors de píxel

Al calcular la diferència de Gaussià obtenim unes coordenades discretes, que refinarem per a cada valor utilitzant l'expansió quadràtica de Taylor de la funció diferència de Gaussià on podrem calcular els seus punts de curvatura.

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D^T}{\partial x^2} x$$

On D i els seus derivats s'avaluen al punt de la mostra i $x = (x, y, \sigma)^T$ és la compensació a partir d'aquest punt. La ubicació de l'extrem, $\bar{x}$, es determina agafant la derivada d'aquesta funció respecte a x i l'igualem a zero.

$$\bar{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

A partir d'aquí igualem les dues funcions anteriors per a obtenir $D(\bar{x})$.

$$D(\bar{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \bar{x}$$

Sabent això podem calcular el determinant $|D(\bar{x})|$, que ens ha de donar un valor entre 0 i 1. Si el valor d'aquest determinant no és superior a 0.03, el punt d'interès en qüestió quedarà automàticament descartat.

Però a partir d'aquí se'ns planteja un problema, i és que el procediment anterior no treballa bé amb aquells punts d'interès que es troben vora les cantonades de les imatges. És per aquest motiu, que per aquests punts s'utilitza la matriu Hessiana, H.

Una matriu Hessiana, és una matriu quadrada, en el nostre cas de 2x2, de la segona derivada d'una funció. Així doncs la funció Hessiana de la nostra funció D és:

$$H = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix}$$

Un cop amb la matriu Hessiana es pot realitzar el determinant i la traça d'aquesta. Per a realitzar-los haurem de tenir en compte uns valors propis, on α serà el valor més gran i on β serà el valor més petit.

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$
$$Det(H) = D_{xx} D_{yy} - (D_{xy})^2 = \alpha \beta$$

En el cas que el determinant sigui negatiu, el punt és descartat per ser un extrem. Considerem r, com el ràtio que hi ha entre la magnitud més gran i la magnitud mínima, per tant α = r β. Llavors;

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

A partir d'aquí podem trobar una relació entre els valors propis i aquest ràtio. Normalment aquest ràtio s'iguala a 10 (r=10), per tant elimina tots aquells punts d'interès que tinguin siguin major a aquesta condició.

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Un cop finalitzat el procediment anterior, obtenim un llistat de punts d'interès que seran representatius de cada imatge, L(x, y). A tots aquests punts, perquè els puguem identificar en altres imatges a diferent escala o amb una rotació diferent, li donarem una orientació, basada en les propietats de la nostra imatge local.

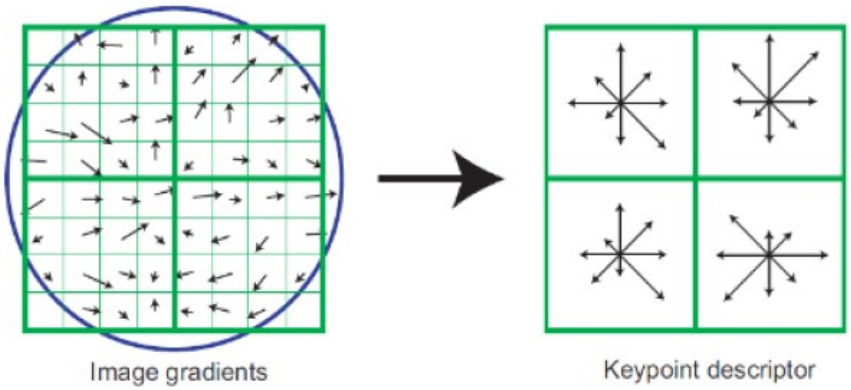
Per a trobar aquesta rotació busquem el gradient, m(x, y), i l'orientació, θ(x, y), utilitzarem la diferència de píxel de la nostra imatge local, L(x, y).

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$$

A partir d'aquí es genera un histograma d'orientacions, format a partir dels gradients d'orientació del conjunt de punts en la regió al voltant de el nostre punt d'interès. Tots aquells punts d'interès que es trobin ala bora de la imatge que no tinguin suficients veïns del píxel seran descartats, i els punts d'interès que no tinguin una orientació dominant també seran descartats. Pot aparèixer el cas que en l'histograma d'un mateix punt d'interès trobem diversos valors d'orientació alts, si els valors són superiors al 80% del valor més alt, en aquell punt d'interès se'n crearà un de nou.

Finalment, amb aquest últim llistat de punts d'interès es calculen els descriptors de cada un d'ells.



Imatge 2.8: A l'esquerra els gradients de la imatge. A la dreta els descriptors dels punts d'interès.

Per al càlcul dels descriptors, primer s'obté la magnitud i l'orientació del gradient en una finestra de 16 píxels per 16 píxels centrada en un punt d'interès. Aquesta finestra la podem dividir en 4 parts iguals, i per cada part es torna a calcular un histograma de 8 direccions diferents amb les orientacions del gradient. Posteriorment, concatenem els histogrames de 8 valors i obtenim un descriptor per a cada punt d'interès de 128 valors.

2.4.2. Algoritme SURF

L'algoritme SURF (Speeded Up Robust Feature), és un algoritme de visió per computador, igual que l'algoritme SIFT, capaç d'obtenir informació detalla i especifica del conjunt d'una imatge. Aquest algoritme creat per Herbert Bay l'any 2006, és un algoritme creat a partir de l'algoritme SIFT, amb algunes modificacions, però pensant més a com optimitzar els processos computacional de l'ordinador, perquè doni millors resultats i d'una manera més ràpida..

A l'igual de l'algoritme SIFT, explicat en l'apartat anterior, el mètode SURF funciona de manera idèntica quant a la detecció de punts.

SURF fa ús de la matriu Hessiana, més concretament, del valor del determinant de la matriu, per la localització i l'escala del punt. El motiu per la utilització de la matriu Hessiana és el fet de la seva rapidesa i precisió. Així doncs donat un punt concret p=(x, y) de la imatge, la matriu H(p, σ) en p a l'escala σ és definida de la següent manera:

$$H(p,\sigma)=\begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix}$$

on $L_{xx}(p, \sigma)$ és la convolució de segon ordre de la Gaussiana $\frac{\partial^2}{\partial p^2} g(\sigma)$ de la imatge I en el punt p, i de manera similar per a $L_{xy}(p, \sigma)$ i per $L_{yy}(p, \sigma)$.

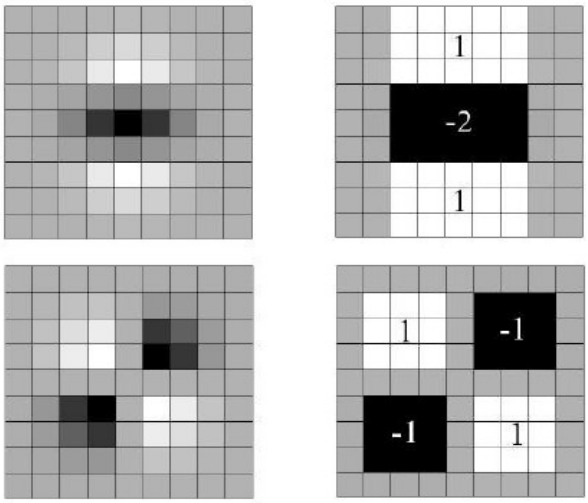
Les aproximacions de les derivades parcials es denoten com Dxx, Dxy i Dyy, i el determinant es calcula de la següent manera.

$$det(H_{aprox.}) = D_{xx}D_{yy} - (w\,D_{xy})^2$$

El pes relatiu, w, de les respostes del filtre s'utilitza per equilibrar l'expressió del determinant Hessià. Això és necessari per a la conservació d'energia entre els nuclis de Gauss i els nuclis gaussians aproximats,

$$w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} = 0.912 \simeq 0.9$$

En la següent imatge podem observar la representació de la derivada parcial de segon ordre d'un filtre Gaussià discretitzat i l'aproximació de la derivada implementada en el cas dels descriptors SURF.

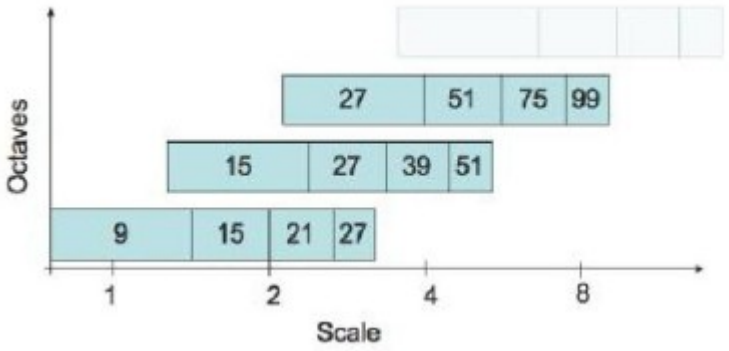


Imatge 2.9: Representació de la derivada parcial de segon ordre en un filtre Gaussià, derivar en y- (Lyy) i direcció xy -(Lxy); i derivada en y- (Dyy) i direcció xy- (Dxy). Les regions en gris són valors iguals a zero.

La imatge de sortida que s'obté després de la convolució de la imatge original amb un filtre de dimensions 9x9, que correspon a la derivada parcial de segon ordre de la gaussiana amb σ =1.2, és considerada com l'escala inicial o també com la màxima resolució espacial.

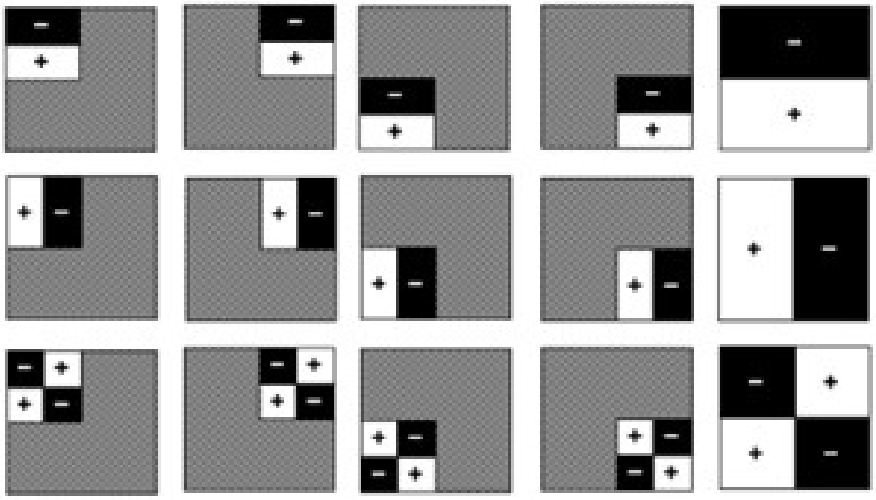
Les capes successives s'obtenen a partir de l'aplicació gradual de filtres de majors dimensions, evitant així els efectes d'aliàsing en la imatge. L'espai escala per als descriptors SURF, igual que en el cas dels descriptors SIFT, està dividit en octaves. A diferència de SIFT, les octaves del SURF estan compostes per un nombre fixes d'imatges resultants. L'increment del filtre dintre d'una octava és el doble respecte del pas de l'octava anterior, i al mateix temps, el primer filtre de cada octava és el segon de l'octava predecessora.

A l'igual de l'algoritme SIFT, a continuació, es filtren tots els punts d'interès obtinguts, amb la condició que ha de ser un valor major que el valor dels seus veïns [Imatge 7].



Imatge 2.10: Funcionament de les octaves en SURF

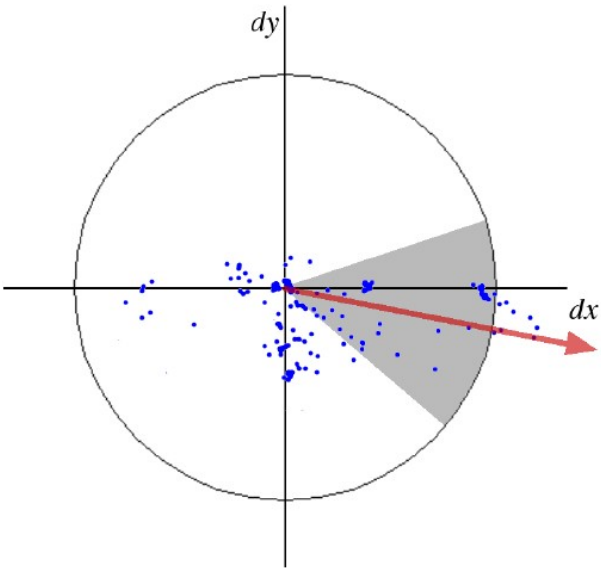
Igual que en l'algoritme SIFT, a partir d'aquest put se li vol donar una orientació a cada punt trobat. Per a trobar l'orientació dels punts es fa servir els filtres de Haar, on els rectangles negres representen les zones amb una contribució positiva al filtre i on els rectangles blancs representen zones amb una contribució negativa al filtre.



Imatge 2.11: Filtres de Haar rotats, traslladats i amb canvis d'escala

Es calcula els veïns més propers posant de radi 6s al voltant del punt d'interès, on s és l'escala a la qual s'han detectat els punts d'interès. Per tal d'anar comparant amb les altres escales s'utilitza el valor de 4s, per anar comprovant les respostes en x i y de cada punt d'interès en cada escala. Totes aquestes imatges a diferent escala queden guardades en memòria fent així que el procés sigui força ràpid. Només calen sis operacions per calcular la resposta en direcció x o y a qualsevol escala.

Un cop calculades i ponderades les respostes amb un gaussià (σ= 2s) centrat en el punt d'interès, les respostes es representen com a punts en un espai amb la resposta horitzontal al llarg de l'abscissa i la resposta vertical al llarg de les ordenades. Finalment, s'obté una orientació dominant per cada sector mitjançant una suma de totes les respostes dintre d'una finestra d'orientació mòbil en un angle de $\frac{\pi}{3}$.



Imatge 2.12: Assignació de l'orientació

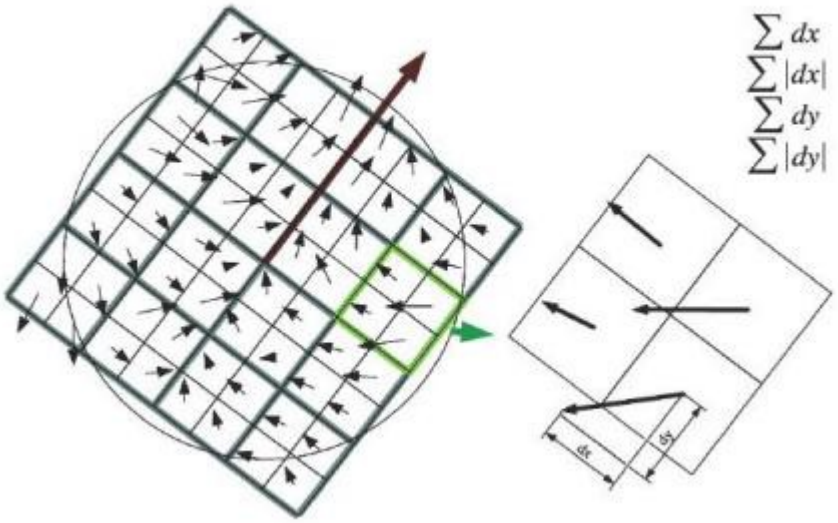
Un cop aquí és calculen els descriptors dels punts. Es construeix una regió quadrada de 20s de grandària al voltant del punt d'interès i orientada amb relació a l'orientació que si li ha donat al punt d'interès. Aquesta regió al seu torn, es divideix en 4x4 subregions. Això permet preservar informació espacial important. Per simplificar, es considera dx i dy a les respostes del filtre Haar en les direccions horitzontals i verticals respectivament.

Per a dotar a les respostes dx i dy d'una robustesa major davant de les deformacions geomètriques i errors de posicionament, aquestes són ponderades per un gaussià de valor σ =3.3s centrat en el punt d'interès. En cada una de les subregions es suma les respostes dx i dy obtenint així un valor de dx i dy representatiu per cada una de les subregions. Al mateix temps es realitza la suma dels valors absoluts de les respostes |dx| i |dy| en cada una de les subregions, obtenint d'aquesta manera, informació de la polaritat sobre els canvis d'intensitat.

Així doncs, cada una de les subregions queda representada per un vector v de components:

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$$

Per tant, si agafem totes les sub-regions, tenim com a resultat un descriptor amb una longitud de 64 valors per cada un dels punts d'interès identificat.



Imatge 2.13: Descriptor resultant del algoritme SURF

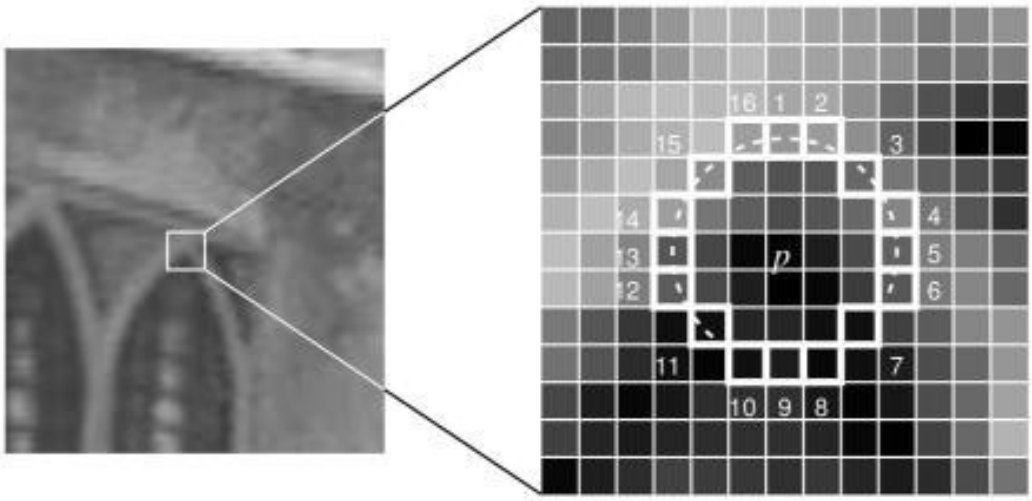
2.4.3. Algoritme ORB

L'algoritme ORB (Oriented FAST and Rotated BRIEF) va ser desenvolupat per els laboratoris OpenCV, per Ethan Rublee, Vincent Rabaud, Kurt Konolige i Gary R. Bradski l'any 2011, com una alternativa més eficient i viable de SIFT i SURF. Orb va ser creat perquè SIFT i SURF són dos algoritmes patentats. ORB, en canvi, és totalment d'ús lliure.

L'algoritme ORB, al igual que SIFT i SURF, el podem dividir en dues parts: una primera part de detecció de punts, que utilitza el detector de punts d'interès FAST (Features from Accelerated Segment Test); i una segona part on orienta els punts trobats creant descriptors, a partir dels descriptors BRIEF (Binary Robust Independent Elementary Features).

La característica principal per la qual s'utilitzen aquests mètodes són per la seva habilitat de poder treballar a baix rendiment, és a dir, de no necessitar un hardware molt potent perquè l'algoritme en si funcioni. ORB és creat per a poder treballar amb imatges de temps real o vídeo, i que pugui funcionar des de un telèfon mòbil. Per aquest motiu, l'algoritme ORB és molt fàcil de fer-lo funcionar però no donarà els resultats més acurats.

A diferència de l'algoritme SIFT i SURF, ORB no treballa amb els veïns més propers per a veure si un punt d'interès és realment això. En aquest cas quan es troba un punt d'interès el píxel seleccionat, anomenat Ip, és crea un cercle de 16 píxels, amb un llindar que anomenem t.



Imatge 2.14: Cercle que es forma al voltant del nostre punt d'interès, Ip.

A partir d'aquí es compara els píxels que queden continguts en aquest cercle (normalment són 16), amb el Ip, per a comparar si són més foscos o clars que el Ip.

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(similar)} \\ b, & I_p + t \le I_{p\to x} & \text{(brighter)} \end{cases}$$

Imatge 2.15: Característiques de llum de píxels

Per a fer l'algorisme més ràpid, primer compara la intensitat dels píxels 1, 5, 9 i 13 del cercle amb el Ip. Si almenys tres dels quatres píxels comparats no són superiors o menors a Ip+t, llavors Ip no és considerant un punt d'interès. Però si al menys 3 dels quatre píxels si que són majors o menors que Ip+t, llavors es farà la comparació amb la resta dels 12 píxels restants. Si els 12 píxels que falten compleixen la condició, els píxels que s'està estudiant ss considera punt d'interès.

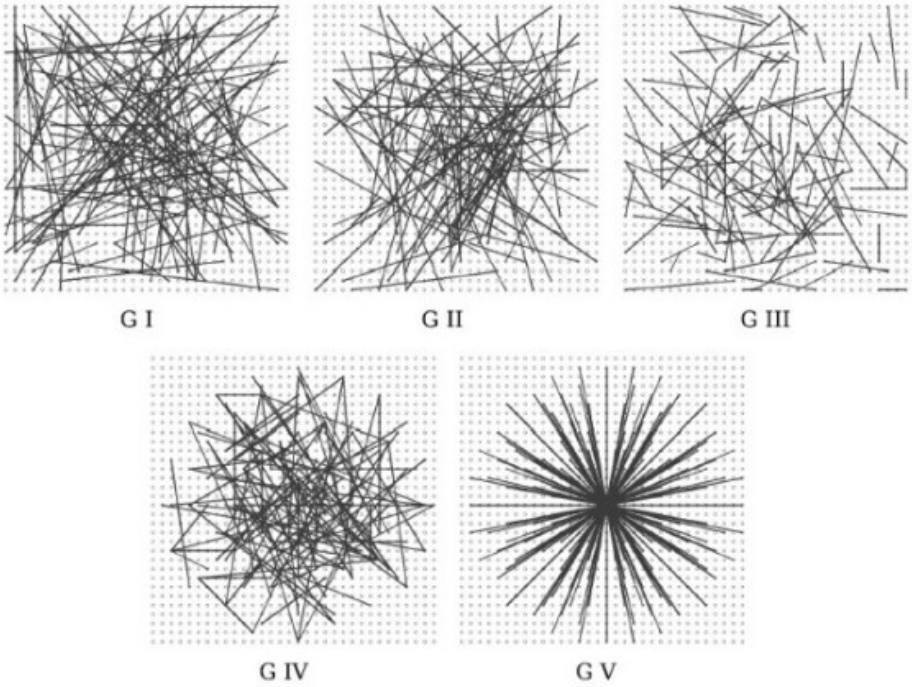
Posteriorment, es crea un descriptor per a cada punt d'interès. Com ja s'ha esmentat, ORB utilitza el mètode BRIEF. Aquest mètode el que fa és suavitzar la imatge a partir de un filtre Gaussià, generant diverses imatges amb un factor de suavitzat cada cop més alt.

Posteriorment es crea un vector característic binari, τ, per la imatge suavitzada, p. Aquest vector binari només conte 1 i 0, i cada punt d'interès està descrit per aquest vector que és de 128-512 bits de llarg. Llavors per τ(p; x, y) és definit com:

$$\tau(p;x,y)=\begin{cases}1 & :p(x)<p(y)\\ 0 & :p(x)\geq p(y)\end{cases}$$

on p(x) és la intensitat de p en el punt x.

Tot aquest procediment genera un n número de vectors, deixant moltes opcions per seleccionar. Aquesta llista de vectors característics binaris de cada punt d'interès serà que ens doni una orientació d'aquest.



Imatge 2.16: Diferents aproximacions de vectors binaris

2.4.4. Cas pràctic de l'algoritme SIFT, SURF i ORB

Per a comprovar quin és l'algoritme més útil, s'ha fet una prova amb 10 imatges, detectant els punts d'interès amb els tres algoritmes. El programa està capat perquè només pugui arribar a generar 1000 punts d'interès com a màxim per imatge.

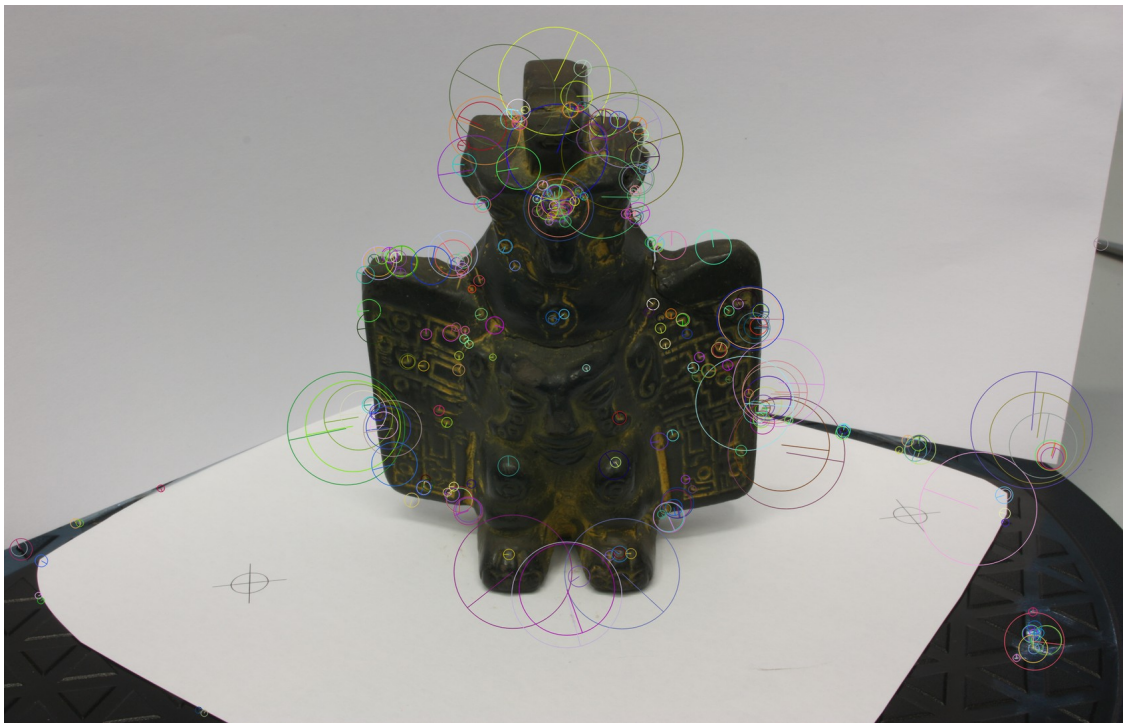
	SIFT		SURF		ORB	
	KeyPoints	Segons	KeyPoints	Segons	KeyPoints	Segons
Imatge 1	750	8	285	2	1000	1
Imatge 2	683	26	280	6	1000	5
Imatge 3	662	43	307	10	1000	9

Imatge 4	648	60	294	14	1000	12
Imatge 5	464	77	243	18	1000	15
Imatge 6	579	94	271	22	1000	19
Imatge 7	513	111	195	26	1000	22
Imatge 8	322	128	155	29	1000	25
Imatge 9	286	145	165	33	968	29
Imatge 10	296	162	140	37	890	32

Taula 2.1: Taula de punts trobats i temps emprats entre els tres algorismes.

Com podem observar en la taula, hi ha una gran diferencia alhora temps emprat en la generació de punts d'interès. En l'algoritme SURF i ORB triguen aproximadament la mateixa quantitat de temps, en canvi l'algoritme SIFT és aproximadament quatre vegades més lent que els dos anteriors. Podem observar a continuació els punts d'interès generats en la primera imatge.

Imatge 2.17: Punts d'interès generats a partir del algoritme SIFT



Imatge 2.18: Punts d'interès generats a partir del algoritme SURF

Imatge 2.19: Punts d'interès generats a partir del algoritme ORB

Com podem observar en les imatges, l'algoritme SIFT és el que troba més punts en la figura, per a una altra banda l'algoritme ORB que és el que troba més punts per imatge, no troba tant punt en la figura sinó que es centra en la cantonada inferior dreta de la imatge.

Així doncs, veient els resultats obtinguts amb els tres algoritmes, emprarem l'algoritme SIFT d'ara endavant.

2.5. Coincidència de punts

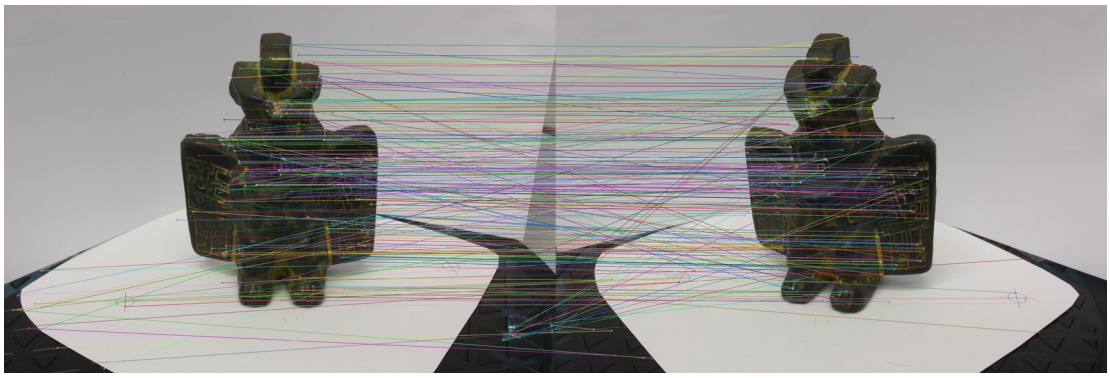
Un cop que ja es tenen els punts d'interès trobats en les diferents imatges, s'han de relacionar entre elles. Aquesta relació es produeix gràcies als descriptors de cada imatge, ja que un punt en la primera imatge (normalment anomenada imatge objecte) que té unes característiques úniques, hauria de trobar-se en la segona (normalment anomenada imatge escena) amb les mateixes característiques. Si es troben dos punts semblants en cada imatge direm que aquests punts són homòlegs.

Un mètode de relació dels punts d'interès de la imatge objecte amb la imatge escena, és l'anomenat de Força Bruta (Brute-Force Matcher). La manera de treballar de la Força Bruta és agafar el descriptor d'un punt d'interès de la imatge objecte i es compara amb tots els descriptors de la imatge escena fent servir els càlculs de distància. Aquells que tinguin la distància menor seran els considerats punts homòlegs.

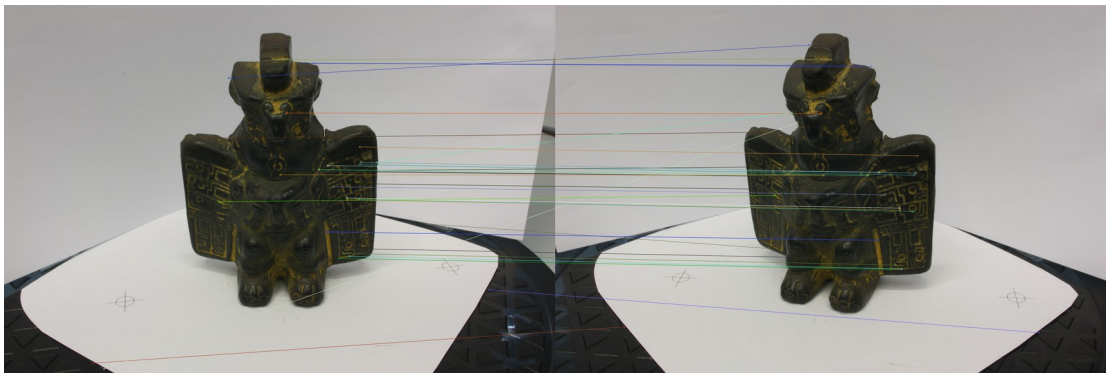
Per una altra banda hi ha el mètode FLANN (Fast Library for Approximate Nearet Neighbors), que és una llibreria que té algoritmes optimitzats per a fer cerques ràpides basades en els veïns més propers.

2.5.1. Cas pràctic de la coincidència de punts

En el cas pràctic s'ha fet servir el mètode de Força Bruta. Després de fer servir el mètode de Força Bruta s'ha realitzat un segon filtratge per a obtenir millor resultats.



Imatge 2.20: Coincidència de punts amb Força Bruta entre imatge 1 i 2



Imatge 2.21: Coincidència de punts amb el segon filtratge entre imatge 1 i 2

2.5.2. Matriu d'homografia

Quan parlem de correspondències entre imatges podem treballar amb la matriu d'homografia, només en aquells casos que les imatges amb les que treballem són imatges d'objectes plans. La matriu d'homografia H, és una matriu de 3x3 de transformació de rang 3, que relaciona els punts de la imatge 1 amb els corresponents de la imatge 2.

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Considerant els punts de la imatge 1 com a x = (x, y, 1) i els de la imatge 2 com a x' = (x', y', 1); podem relacionar-los amb la matriu d'homografia.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

La matriu d'homografia es fa servir per a figures planes i s'utilitza bàsicament per alinear les imatges, compensació de moviment, i és molt important per als sistemes de reconeixement d'objectes.

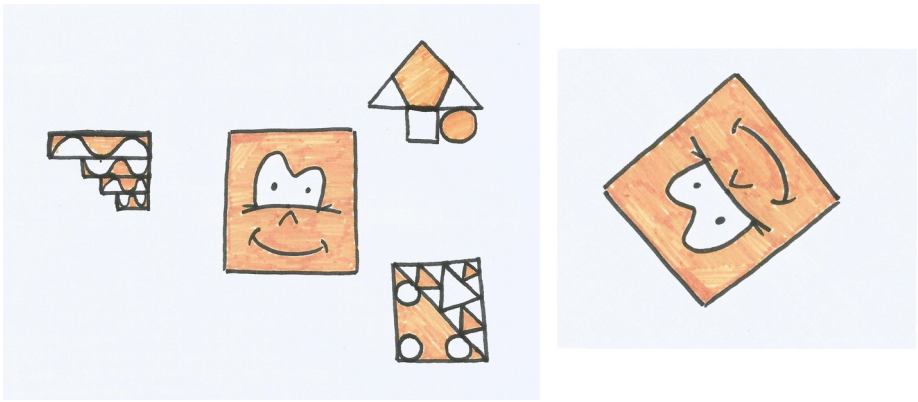
Gràcies a la matriu d'homografia també podem veure si els punts corresponents difereixen molt els uns dels altres calculant el seu error i veient si la distancia entre els punts de la primera imatge i els de la segona és molt gran. Si aquesta distància supera certa quantitat (per exemple 1 píxel), els punts queden descartats automàticament.

$$\sum_i \left(x'_i - \frac{h_{11} x_i + h_{12} y_i + h_{13}}{h_{31} x_i + h_{32} y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21} x_i + h_{22} y_i + h_{23}}{h_{31} x_i + h_{32} y_i + h_{33}} \right)^2$$

2.5.2.1. Cas pràctic d'homografia

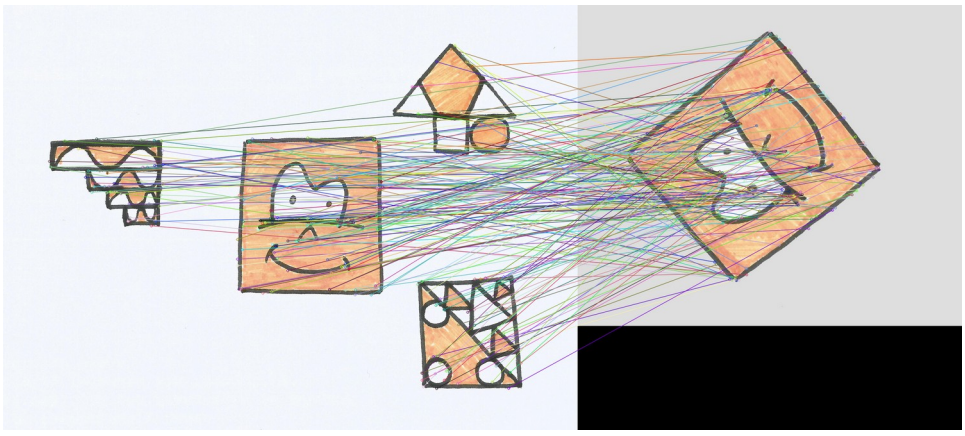
Al llarg de la realització d'aquest treball s'ha volgut fer un petit experiment amb la matriu d'homografia. S'ha realitzat una composició de dibuixos a mà alçada, on posteriorment s'aïllarà un dels dibuixos i se li aplicarà un canvi de mida i una rotació. A partir d'aquí volem observar el fet que si realment la matriu d'homografia ens ajuda a trobar el dibuix modificat en el conjunt de dibuixos.

Els dibuixos utilitzats es van realitzar a partir de figures geomètriques no molt complexes perquè no fossin molt difícils de reconèixer, i es va fer servir un color perquè els punts característics fossin més fàcils de calcular [Imatge 2.22].



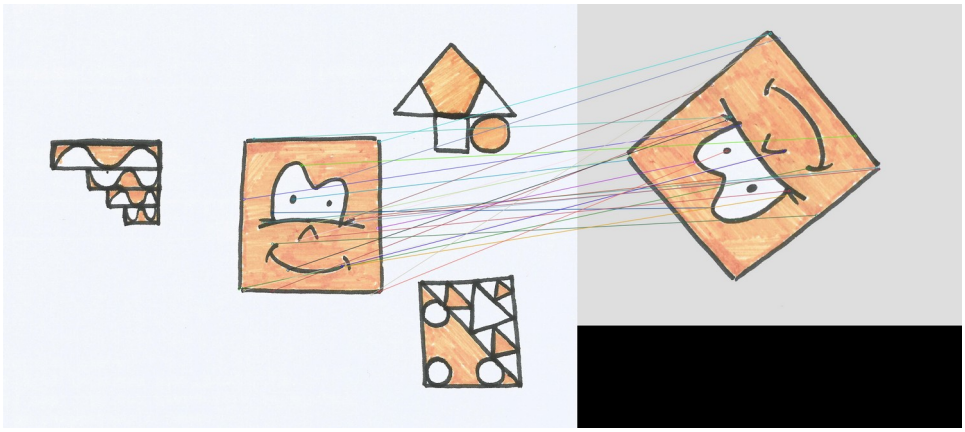
Imatge 2.22: A la esquerra conjunt de dibuixos, a la dreta dibuix a trobar

A partir d'aquí és va crear un nou mòdul de treball, on es van carregar les imatges i es van calcular els seus punts d'interès. Posteriorment es va realitzar la coincidència de punts entre les dues imatges.

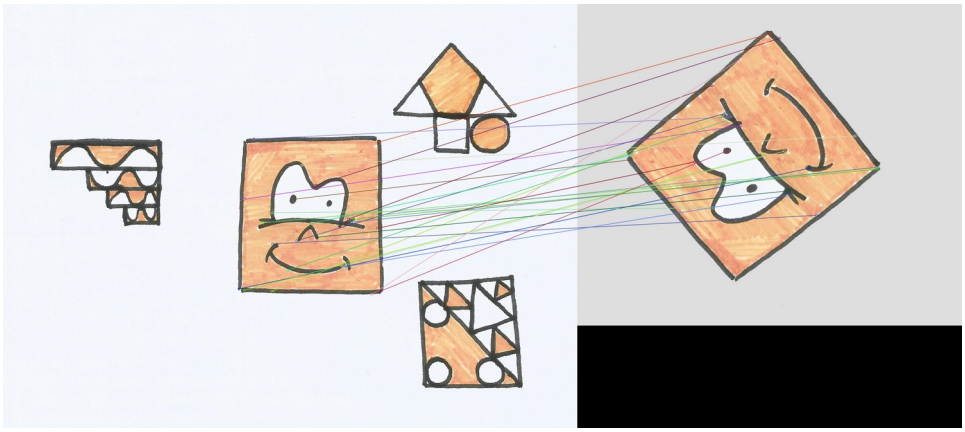


Imatge 2.23: Coincidència de punts entre les dues imatges sense filtratge

Al realitzar la coincidència de punts entre les imatges obtenim 221 coincidències entre la imatge 1 i la imatge 2 (el canvi de color en el fons de la imatge 2 és purament visual, no afecta cap dels resultats). Com podem observar en la imatge resultant [Imatge 23], no totes les coincidències que tenim són «bones», ja que hi han moltes que són causades per altres figures que no pas per la que ens interessa. És en aquest moment en el qual apliquem el filtre a partir de la matriu d'homografia. Com ja s'ha explicat en l'apartat anterior, amb la matriu d'homografia i els punts de cada imatge es calcula un error, en aquest experiment considerem que si l'error de dos punts superen la mesura d'un píxel seran eliminats. Després del primer filtratge amb la matriu d'homografia s'ha tornat a calcular una segona matriu per veure si en una segona passada amb les mateixes condicions de filtratge pot arribar a depurar encara més punts.



Imatge 2.24: Coincidències calculant l'error amb la matriu d'homografia



Imatge 2.25: Coincidències calculant l'error amb la segona matriu d'homografia

Com podem observar la depuració que es pot realitzar amb la matriu d'homografia ens ajuda a eliminar aquelles coincidències que relacionaven la nostra figura amb la resta del conjunt. També ens podem fixar que encara que el dibuix quedi aïllat moltes de les coincidències que tenim no són correctes, ja que ha emparellat punts d'interès que no corresponen entre la imatge 1 i 2. Es pot observar el codi de l'experiment en els annexos (pàgina 56).

2.5.3. Matriu fonamental

La geometria epipolar és la geometria projectiva intrínseca entre dos punts de vista. És independent de l'estructura de la imatge, i només depèn dels paràmetres interns de les càmeres i de la seva posició relativa.

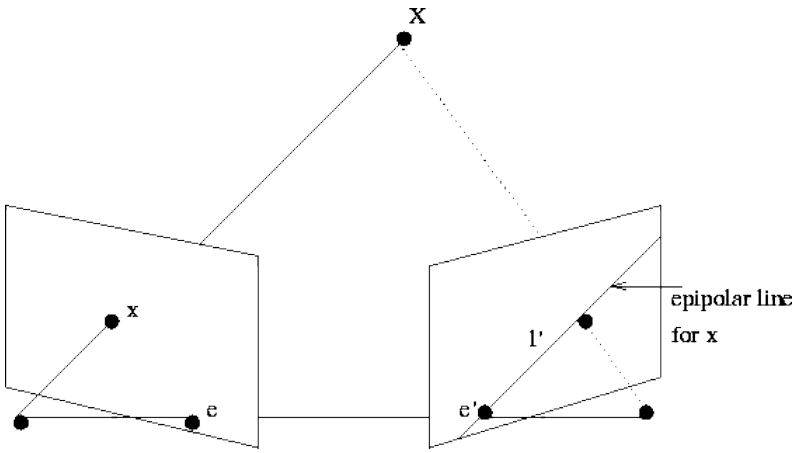
La matriu fonamental F encapsula aquesta geometria intrínseca. És una matriu 3x3 de rang 2. Si un punt en l'espai és imaginat en com a x en la primera imatge, i com a x' en la segona, llavors els punts de les imatges satisfà la relació $x'^{T} F x = 0$.

$$F = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix}$$

A diferència del que ens trobem en la matriu d'homografia, la matriu fonamental ens serveix per a la relació d'imatges que tenen profunditat.

La matriu fonamental té un seguit de característiques úniques:

- Transposada; si F és la matriu fonamental de dos parells de càmeres (P, P'), llavors F^{T} és la matriu fonamental de la parella en l'ordre oposat (P', P).
- Línies epipolars: per a qualsevol punt x en la primera imatge, la seva línia epipolar corresponent és $l' = Fx$. De la mateixa manera, $l = F^{T}x'$ representa la línia epipolar corresponent a x' en la segona imatge.



Imatge 2.26: Geometria epipolar amb dos parells d'imatges

- L'epipol: per a qualsevol punt x (que no sigui e, on e és punt d'intersecció de la línia que uneix els centres de les càmeres amb el pla de la imatge), la línia epipolar l' = Fx conte l'epipole e'. Així doncs e' satisfeis $e'^{T}(Fx) = (e'^{T}F)x = 0$ per tots les x.
- F té set graus de llibertat: una matriu homogènia de 3x3 té vuit ràtios independents (hi ha nou elements, i el valor escalar que no és significant); per una altra banda, F també satisfà el traçat del *det*F = 0 el qual elimina un grau de llibertat.

2.5.3.1. Cas pràctic de matriu fonamental

Igual que en el punt anterior on s'ha tractat l'homografia, també s'ha volgut realitzar un petit experiment amb la matriu fonamental. En el moment de càlcul de la matriu fonamental amb la funció que ens proporciona la llibreria de OpenCV, aquesta utilitza l'algoritme RANSAC. Bàsicament, l'algoritme RANSAC agafa tots els punts d'introduïts per al càlcul de la matriu fonamental i tots aquells punts que tenen valors atípics a la resta de valors no els té en compte. Això vol dir que si tenim dos llistats de 20 punts cadascun, on d'aquests totals 3 són atípics, en el moment de calcular la matriu fonamental que relacioni els dos llistats, l'algorisme RANSAC no els tindrà presents; però això no significa que aquests punts que es poden considerar atípics quedin descartats de les nostres llistes. És per aquest, que ens hem vist obligats a aplicar un filtre a aquestes llistes, per a trobar gràcies a la condició de coplanaritat x'T F x = 0, quins són aquells que l'algoritme RANSAC no ha tingut en compte, i poder-los eliminar.

Per a realitzar aquest filtratge, s'ha seguit la condició de coplanaritat:

$$S = \vec{x'}^{T} \, F \, \vec{x} = 0$$

On,

$$\vec{x'} = \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} \qquad \vec{x} = \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} i$$

Sent u1 i v1 les coordenades imatges mesurades sobre el primer fotograma i u2 i v2 les coordenades imatges mesurades sobre el segon fotograma.

A partir d'aquí comprovar com afecta el càlcul de S un error, per exemple, d'un píxel sobre les coordenades imatges. Per aquest motiu es realitza el càlcul diferencial sobre S respecte a les coordenades imatge:

$$dS = \frac{\partial S}{\partial u_1}\, du_1 + \frac{\partial S}{\partial v_1}\, dv_1 + \frac{\partial S}{\partial u_2}\, du_2 + \frac{\partial S}{\partial v_2}\, dv_2$$

Si desenvolupem la condició de coplanaritat:

$$S = \vec{x'}^{T}\, F\, \vec{x} = \begin{pmatrix} u_1 & v_1 & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} =$$

$$\begin{pmatrix} u_1 & v_1 & 1 \end{pmatrix} \begin{pmatrix} f_{11}\, u_2 + f_{12}\, v_2 + f_{13} \\ f_{21}\, u_2 + f_{22}\, v_2 + f_{23} \\ f_{31}\, u_2 + f_{32}\, v_2 + f_{33} \end{pmatrix} =$$

$$f_{11}\, u_1\, u_2 + f_{12}\, u_1\, v_2 + f_{13}\, u_1 + f_{21}\, v_1\, u_2 + f_{22}\, v_1\, v_2 + f_{23}\, v_1 + f_{31}\, u_2 + f_{32}\, v_2 + f_{33}$$

A partir d'aquí podem realitzar el càlcul diferencial:

$$dS = \frac{\partial S}{\partial u_1}\, du_1 + \frac{\partial S}{\partial v_1}\, dv_1 + \frac{\partial S}{\partial u_2}\, du_2 + \frac{\partial S}{\partial v_2}\, dv_2$$

$$\frac{\partial S}{\partial u_1} = f_{11}\, u_2 + f_{12}\, u_2 + f_{13}$$

$$\frac{\partial S}{\partial v_1} = f_{21}\, u_2 + f_{22}\, v_2 + f_{23}$$

$$\frac{\partial S}{\partial u_2} = f_{11}\, u_1 + f_{21}\, v_1 + f_{31}$$

$$\frac{\partial S}{\partial v_2} = f_{12}\, u_1 + f_{22}\, v_1 + f_{32}$$

On el resultat final serà:

$$dS = \left(f_{11}\,u_2 + f_{12}\,u_2 + f_{13}\right) du_1 + \left(f_{21}\,u_2 + f_{22}\,v_2 + f_{23}\right) dv_1 + \left(f_{11}\,u_1 + f_{21}\,v_1 + f_{31}\right) du_2 + \left(f_{12}\,u_1 + f_{22}\,v_1 + f_{32}\right) dv_2$$

A partir d'aquí s'han plantejat 3 casos diferents amb un conjunt de dades sintètiques.

- En el primer cas s'aplica un truncament a les dades, deixant-les com a enters. En aquest cas s'espera que l'error obtingut sigui al voltant del píxel. La condició que se li dóna al programa és que si el producte S supera 3 vegades el diferenciat estimat per a la variació d'un píxel aquest punt sigui dolent, en aquest cas cap de les dades hauria de ser considerada com a dolenta.

```
//////////////////
COORDENADES TEST 1
//////////////////
Fonamental=
[2.042169198655077e-10, 1.432247947341257e-07, -2.409565818339843e-05;
-1.196661798563318e-07, 1.893907801464893e-08, 0.005658161793867134;
-8.673222671701231e-06, -0.005851112472583131, 1]

S( u1, v1, u2, v2) = 0
dS( u1, v1, u2, v2) = 0
Punt 1:     Punt bo
 S( 6729, 4035, 7722, 4133) = 1.24345e-13
 dS( 6729, 4035, 7722, 4133) = 0.0106829

Punt 2:     Punt bo
 S( 8419, 500, 9455, 646) = -2.25597e-13
 dS( 8419, 500, 9455, 646) = 0.00931193

Punt 3:     Punt bo
 S( 8200, 8619, 9192, 8868) = 4.76064e-13
 dS( 8200, 8619, 9192, 8868) = 0.0115259

Punt 4:     Punt bo
 S( 6888, 8589, 7822, 8789) = 0.0104821
 dS( 6888, 8589, 7822, 8789) = 0.0118619

Punt 5:     Punt bo
 S( 2263, 1723, 3343, 1816) = -3.94191e-06
 dS( 2263, 1723, 3343, 1816) = 0.011238

Punt 6:     Punt bo
 S( 6642, 5832, 7597, 5945) = 0.00812678
 dS( 6642, 5832, 7597, 5945) = 0.0111851

Punt 7:     Punt bo
 S( 3757, 3291, 4768, 3355) = -0.00101269
 dS( 3757, 3291, 4768, 3355) = 0.0112609

Punt 8:     Punt bo
 S( 1355, 963, 2502, 1083) = 4.52971e-14
 dS( 1355, 963, 2502, 1083) = 0.0112732

Punt 9:     Punt bo
 S( 5613, 7496, 6539, 7617) = 0.000420461
 dS( 5613, 7496, 6539, 7617) = 0.0118979

Punt 10:     Punt bo
 S( 3156, 1046, 4240, 1167) = 1.59872e-14
 dS( 3156, 1046, 4240, 1167) = 0.0108293

Punt 11:     Punt bo
 S( 3437, 9058, 4309, 9150) = 6.60805e-13
 dS( 3437, 9058, 4309, 9150) = 0.0128823

Punt 12:     Punt bo
 S( 4235, 4601, 5208, 4658) = -0.000197927
 dS( 4235, 4601, 5208, 4658) = 0.0114831

Punt 13:     Punt bo
 S( 5961, 40, 6991, 200) = -0.00221368
 dS( 5961, 40, 6991, 200) = 0.00984017

Punt 14:     Punt bo
 S( 2471, 6430, 3417, 6449) = -0.000815439
 dS( 2471, 6430, 3417, 6449) = 0.0124247

Punt 15:     Punt bo
 S( 5103, 2769, 6109, 2856) = 7.81597e-14
 dS( 5103, 2769, 6109, 2856) = 0.0107742
```

Imatge 2.27: Resultat del cas pràctic de matriu fonamental, coordenades truncades

Després de realitzar el càlcul podem observar que cap error supera el píxel estimat, i totes les dades són bones.

- En el segon cas les coordenades en píxel se li han afegit un decimal. En aquest cas s'espera que l'error obtingut d'1/10 de píxel.

Igual que en el cas anterior ens fem ajuda de la llibreria OpenCV per a realitzar els càlculs.

```
//////////////////
COORDENADES TEST 2
//////////////////
Fonamental=
[9.031400159181014e-11, 2.021905681652612e-08, 1.109719828701339e-05;
3.021964344069297e-09, 2.328468122258953e-08, 0.005413084332100304;
-2.345562946523549e-05, -0.005754129428630184, 1]

S( u1, v1, u2, v2) = 0
dS( u1, v1, u2, v2) = 0
Punt 1:     Punt bo
 S( 6729.7, 4035.2, 7722.7, 4133.7) = 0.000298582
 dS( 6729.7, 4035.2, 7722.7, 4133.7) = 0.0111628

Punt 2:     Punt bo
 S( 8419.9, 500, 9455.6, 646) = 1.96732e-13
 dS( 8419.9, 500, 9455.6, 646) = 0.0110751

Punt 3:     Punt bo
 S( 8200.3, 8619.1, 9192.6, 8868) = -7.74492e-13
 dS( 8200.3, 8619.1, 9192.6, 8868) = 0.0112295

Punt 4:     Punt bo
 S( 6888.4, 8589.3, 7822.8, 8789.8) = -9.09495e-13
 dS( 6888.4, 8589.3, 7822.8, 8789.8) = 0.0112489

Punt 5:     Punt bo
 S( 2263.5, 1723.5, 3343.1, 1816.1) = 0.000173501
 dS( 2263.5, 1723.5, 3343.1, 1816.1) = 0.0111999

Punt 6:     Punt bo
 S( 6642.1, 5832.8, 7597, 5945.9) = -6.46594e-13
 dS( 6642.1, 5832.8, 7597, 5945.9) = 0.0111957

Punt 7:     Punt bo
 S( 3757.9, 3291.3, 4768, 3355.4) = -0.000318583
 dS( 3757.9, 3291.3, 4768, 3355.4) = 0.0111997

Punt 8:     Punt bo
 S( 1355.1, 963.2, 2502.2, 1083) = 5.96465e-05
 dS( 1355.1, 963.2, 2502.2, 1083) = 0.0112038

Punt 9:     Punt bo
 S( 5613.2, 7496.3, 6539.5, 7617.6) = -9.30811e-13
 dS( 5613.2, 7496.3, 6539.5, 7617.6) = 0.0112423

Punt 10:     Punt bo
 S( 3156.2, 1046.4, 4240.6, 1167.4) = -1.42109e-13
 dS( 3156.2, 1046.4, 4240.6, 1167.4) = 0.0111741

Punt 11:     Punt bo
 S( 3437.9, 9058.2, 4309.9, 9150.7) = 0.000979579
 dS( 3437.9, 9058.2, 4309.9, 9150.7) = 0.0113136

Punt 12:     Punt bo
 S( 4235.4, 4601.3, 5208.3, 4658.3) = 2.09825e-05
 dS( 4235.4, 4601.3, 5208.3, 4658.3) = 0.0112136

Punt 13:     Punt bo
 S( 5961, 40.7, 6991.6, 200.1) = 1.5099e-13
 dS( 5961, 40.7, 6991.6, 200.1) = 0.0111101

Punt 14:     Punt bo
 S( 2471.5, 6430, 3417.7, 6449) = 3.4627e-08
 dS( 2471.5, 6430, 3417.7, 6449) = 0.0112736

Punt 15:     Punt bo
 S( 5103.3, 2769.2, 6109.3, 2856.4) = 8.96403e-06
 dS( 5103.3, 2769.2, 6109.3, 2856.4) = 0.0111686
```

Imatge 2.28: Resultats cas pràctic de matriu fonamental, coordenades amb un decimal

- En el tercer cas, s'han realitzat canvis grollers, de 100 píxels, en 4 de les 15 coordenades. S'espera que d'aquests canvis en el moment de realitzar el calcul de la matriu fonamental l'algorisme RANSAC hauria d'eliminar aquests punts, i la matriu fonamental hauria de ser molt semblant a la del primer cas [Imatge 2.27].

```
//////////////////
COORDENADES TEST 3
//////////////////
Fonamental=
[-5.520873191783456e-09, 1.987738558109249e-06, -0.001673036080989926;
-1.92554969580311e-06, -7.941854573259146e-09, 0.01836159355877054;
0.001432252002202712, -0.01687776890914172, 1]

S( u1, v1, u2, v2) = 0
dS( u1, v1, u2, v2) = 0
Punt 1:      Punt bo
 S( 6729, 4035, 7722, 4133) = -1.49214e-13
 dS( 6729, 4035, 7722, 4133) = 0.0198681

Punt 2:      Punt bo
 S( 8419, 500, 9455, 646) = -2.94431e-13
 dS( 8419, 500, 9455, 646) = 0.00116151

Punt 3:      Punt bo
 S( 8200, 8619, 9192, 8868) = -1.42109e-13
 dS( 8200, 8619, 9192, 8868) = 0.0323511

Punt 4:      Punt dolent
 S( 6888, 8589, 7822, 8989) = -0.529179
 dS( 6888, 8589, 7822, 8989) = 0.0377789

Punt 5:      Punt bo
 S( 2263, 1723, 3343, 1816) = -1.06581e-14
 dS( 2263, 1723, 3343, 1816) = 0.0281195

Punt 6:      Punt dolent
 S( 6642, 5832, 7597, 5945) = 0.137438
 dS( 6642, 5832, 7597, 5945) = 0.0273439

Punt 7:      Punt bo
 S( 3757, 3291, 4768, 3355) = 6.26152e-05
 dS( 3757, 3291, 4768, 3355) = 0.0284849

Punt 8:      Punt dolent
 S( 1355, 963, 2502, 1283) = -2.86782
 dS( 1355, 963, 2502, 1283) = 0.0290187

Punt 9:      Punt bo
 S( 5613, 7496, 6539, 7617) = -2.84217e-14
 dS( 5613, 7496, 6539, 7617) = 0.0379542

Punt 10:      Punt bo
 S( 3156, 1046, 4240, 1167) = -8.88178e-15
 dS( 3156, 1046, 4240, 1167) = 0.0220233

Punt 11:      Punt bo
 S( 3437, 9058, 4309, 9150) = -0.0753975
 dS( 3437, 9058, 4309, 9150) = 0.0526289

Punt 12:      Punt dolent
 S( 4235, 4601, 5208, 4858) = -1.68097
 dS( 4235, 4601, 5208, 4858) = 0.0321962

Punt 13:      Punt bo
 S( 5961, 40, 6991, 200) = -1.13687e-13
 dS( 5961, 40, 6991, 200) = 0.0125641

Punt 14:      Punt bo
 S( 2471, 6430, 3417, 6449) = -0.0269758
 dS( 2471, 6430, 3417, 6449) = 0.0458376

Punt 15:      Punt bo
 S( 5103, 2769, 6109, 2856) = 0.0149354
 dS( 5103, 2769, 6109, 2856) = 0.02123
```

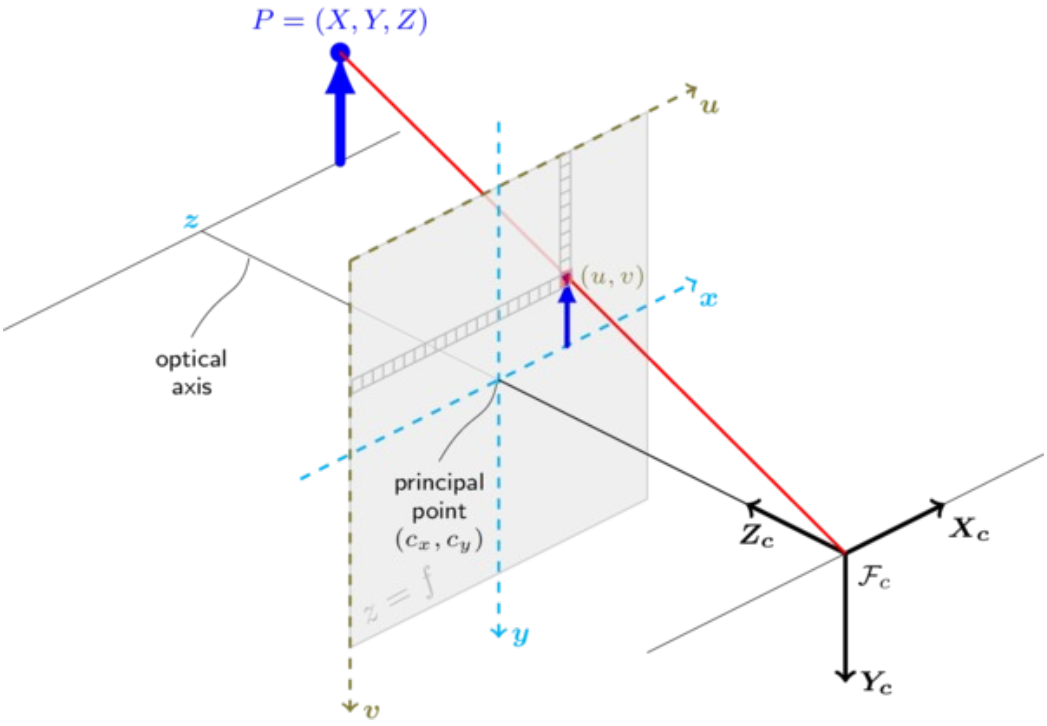
Imatge 2.29: Resultats cas pràctic de matriu fonamental, coordenades amb errors de 100 píxels

Com podem observar, el programa ha detectat els punts 4, 6, 8 i 12, que són els quatre punts als quals se'ls hi havia augmentat el valor en la coordenada y per 100 píxels.

El codi d'aquest experiment el podem trobar en els annexos (pàgina 62).

2.6. Reconstrucció 3D

Després de la identificació dels punts d'interès entre les diferents seqüències d'imatges, podem observar que hi ha punts homòlegs entre elles, aquests punts homòlegs seran els que es transformaran en punts 3D per la reconstrucció del nostre model.



Imatge 2.30: Model de càmera estenopeica

Abans, però, de poder realitzar aquest procediment s'han de tenir en compte uns paràmetres necessaris per a la reconstrucció.

Un d'aquests paràmetres serien els paràmetres intrínsecs de la càmera. Es tracta d'una matriu de 3x3 que ens descriu els paràmetres interns de cada càmera. En aquesta matriu trobarem la distància focal de la càmera en unitat de píxels i el punt principal de la nostra imatge, que normalment és el centre de la imatge.

$$K = \begin{pmatrix} f_{píxel} & 0 & c_x \\ 0 & f_{píxel} & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Aquesta matriu és invariant al llarg de tot el procés a no ser que es canviï la distància focal.

Per una altra banda tenim els paràmetres extrínsecs de càmera, una matriu 3x4 corresponents a la rotació i a la translació de les imatges, s'utilitza per descriure el moviment de la càmera al voltant d'una escena estàtica, o viceversa, un moviment rígid d'un objecte davant d'una càmera fixa.

$$[R||t] = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}$$

Aquests paràmetres els podem relacionar en la següent funció per a trobar els punts.

$$s\, m' = K\, [R||t]\, M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_{píxel} & 0 & c_x \\ 0 & f_{píxel} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Però el problema d'aquesta funció és que no podem trobar les coordenades X, Y i Z, ja que no es pot fer la inversa de la matriu [R||t] perquè no és una matriu quadrada. Però això canvia quan z ≠ 0.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x / z$$
$$y' = y / z$$
$$u = f \cdot x' + c_x$$
$$v = f \cdot y' + c_y$$

Perquè funcioni s'afegeix un factor d'escala (s) i la matriu intrínseca de càmera (K), per arribar a u, v per a trobar les coordenades X, Y i Z.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \left(s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} A^{-1}\, t \right) R^{-1}$$

2.6.1. Cas pràctic de la reconstrucció 3D

Quan treballem amb la biblioteca OpenCV, tota la teoria explicada en el punt anterior queda encapsulada en una sola funció, *cv::sfm::reconstruct()*. Aquesta funció agafa les coordenades píxel de la imatge, juntament amb la matriu de càmera, per a donar com a resultat un llistat de punts amb les coordenades 3D.

En el moment abans de fer servir aquesta funció és va voler realitzar un experiment amb unes dades sintètiques per a comprovar que aquesta funció donaria els resultats esperats. Per a crear aquestes dades sintètiques es van agafar un llistat de 30 punts terrenys coneguts, d'on es van crear 4 fotogrames diferents, fent servir la condició de col·linealitat:

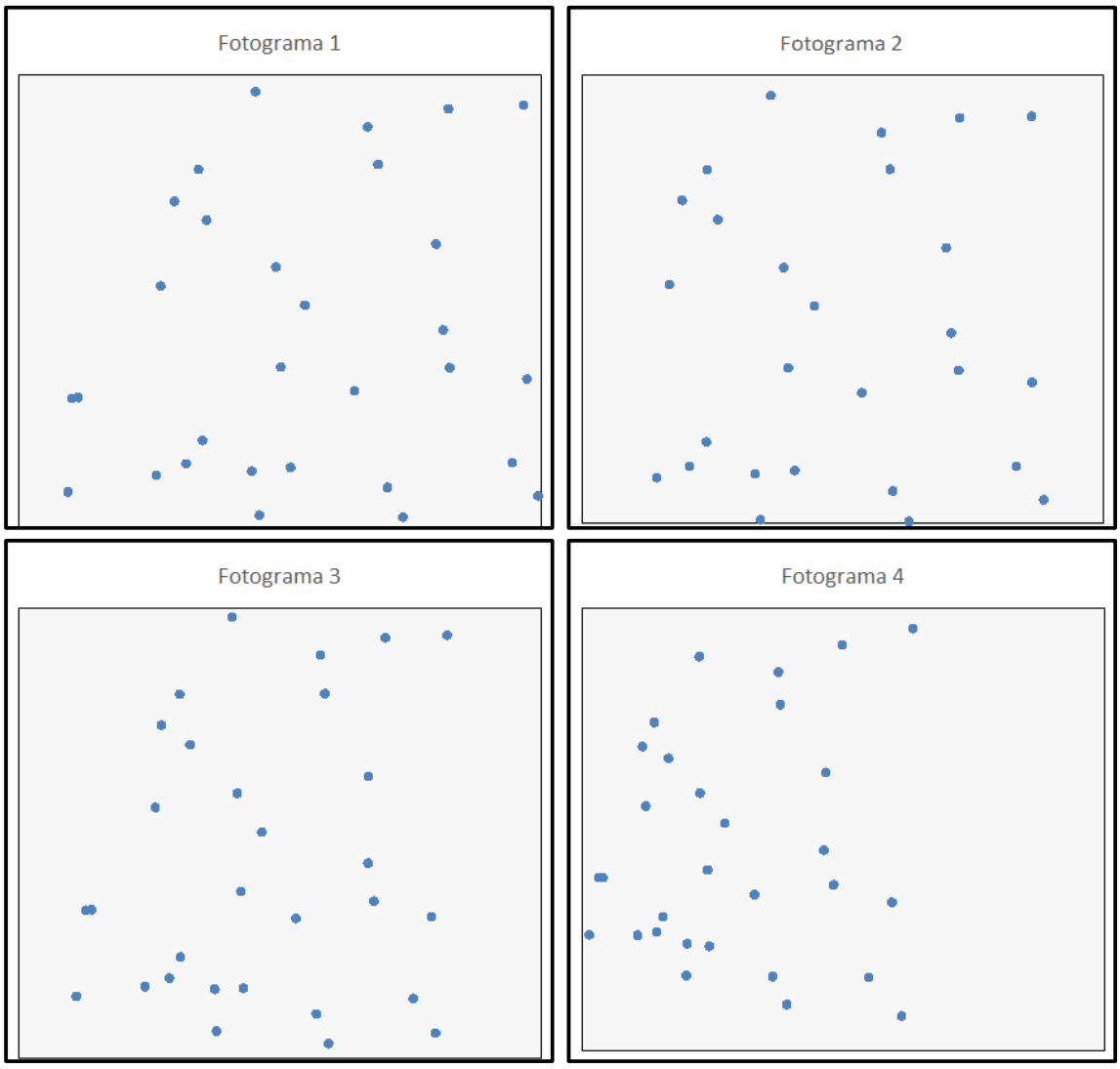
$$x' = -f\,\frac{r_{11}(X-X_0)+r_{12}(Y-Y_0)+r_{13}(Z-Z_0)}{r_{31}(X-X_0)+r_{32}(Y-Y_0)+r_{33}(Z-Z_0)}$$

$$y' = -f\,\frac{r_{21}(X-X_0)+r_{22}(Y-Y_0)+r_{23}(Z-Z_0)}{r_{31}(X-X_0)+r_{32}(Y-Y_0)+r_{33}(Z-Z_0)}$$

En tots i cada un dels fotogrames s'han utilitzat diferents incògnites per a donar-los un punt de vista diferent:

$FOTOGRAMA\,1$	$FOTOGRAMA\,2$	$FOTOGRAMA\,3$	$FOTOGRAMA\,4$
$\omega = 0^o$	$\omega = 1^o$	$\omega = -1^o$	$\omega = -1^o$
$\varphi = 0^o$	$\varphi = -1^o$	$\varphi = 2^o$	$\varphi = 15^o$
$\kappa = 0^o$	$\kappa = 1^o$	$\kappa = 3^o$	$\kappa = 3^o$
$X_0 = 1000\,m$	$X_0 = 1100\,m$	$X_0 = 1200\,m$	$X_0 = 1500\,m$
$Y_0 = 2000\,m$	$Y_0 = 2000\,m$	$Y_0 = 2005\,m$	$Y_0 = 2005\,m$
$Z_0 = 500\,m$	$Z_0 = 500\,m$	$Z_0 = 500\,m$	$Z_0 = 500\,m$

On a partir dels angles ω, φ i κ podem aconseguir la matriu de rotació i així poder fer servir la condició de col·linealitat. També a cada foto coordenada se li ha sumat un soroll gaussià amb una desviació estàndard de 12 μm.



Imatge 2.31: Fotogrames de les dades sintètiques

Posteriorment s'ha convertit les foto coordenades de mil·límetres a coordenades píxel, on s'ha considerat un sensor de càmera de 12 x 12 mil·límetres, i una mida d'imatge de 9500 x 9500 píxels.

En aquest punt s'han carregar la llista de punts obtinguts per un programa amb les llibreries OpenCV instal·lades, i s'han generat coordenades terreny amb l'algoritme *cv::sfm::reconstruct()*.

Un cop obtingudes les coordenades que ens proposa el programa s'ha calculat un ajust en bloc de les coordenades calculades amb el mètode dels mínims quadrats. Un cop realitzat l'ajust en bloc s'ha fet una comparativa entre les coordenades calculades i les coordenades originals i s'ha pogut comprovar que l'error per cada punt.

PUNT	X original	X calculat	Error (m)	PUNT	X original	X calculat	Error (m)
P1	1278,000	1278,0	-0,019	P16	751,000	750,9	0,062
P2	1470,000	1469,7	0,276	P17	551,000	550,9	0,131
P3	1442,000	1442,0	-0,016	P18	700,000	699,8	0,202
P4	1293,000	1293,1	-0,142	P19	1165,000	1165,0	0,025
P5	790,000	789,9	0,082	P20	1437,000	1436,6	0,358
P6	1269,000	1269,1	-0,115	P21	1401,000	1400,6	0,427
P7	951,000	951,1	-0,050	P22	1300,000	1300,2	-0,150
P8	698,000	698,1	-0,095	P23	800,000	799,6	0,357
P9	1148,000	1148,0	-0,042	P24	1130,000	1130,2	-0,239
P10	893,000	893,2	-0,160	P25	531,000	530,4	0,637
P11	900,000	899,8	0,219	P26	944,000	943,8	0,189
P12	1000,000	1000,0	0,041	P27	537,000	536,6	0,367
P13	1200,000	1200,1	-0,079	P28	911,000	911,0	0,019
P14	800,000	799,9	0,066	P29	752,000	751,8	0,211
P15	1100,000	1100,0	-0,027	P30	971,000	971,0	0,047

Taula 2.2: Comparativa entre coordenades originals i calculades

Com podem observar l'error obtingut no supera, en el pitjor dels casos, els 50 centímetres. Tenint en compte les condicions inicials que s'han indicat al principi de l'experiment, les dades obtingudes es poden tolerar com a bones, indicant d'aquesta manera que la funció *cv::sfm::reconstruct()* de la llibreria OpenCV és solvent en aquests tipus de problemes.

2.7. Funcionament del programa

Quan es va plantejar el com havia de ser el programa d'aquest projecte hi havia un objectiu clar: que funciones per a més d'una imatge carregada. La majoria dels exemples que podem trobar amb OpenCV on es treballa amb imatges, són exemples on només es carreguen dues imatges i es realitzen tot un seguit d'operacions sobre aquestes. En aquest projecte en canvi s'ha creat perquè es puguin carregar tantes imatges com es vulguin i que sigui un procés semi-automàtic.

El projecte es pot dividir en dos grans blocs:

- Bloc groc. Aquest primer bloc conté el mòdul 1 (pàgina 45) i el mòdul 2 (pàgina 46). En aquest primer bloc l'objectiu final és aconseguir un fitxer de text, on estaran escrites les direccions de les imatges que es volen treballar.

- Bloc verd. Diferenciem aquest mòdul de l'anterior, ja que pel funcionament d'aquest han d'estar instal·lades les llibreries de OpenCV. El mòdul 3 (pàgina 48), és el mòdul que més documents genera:
  - Llista de punts d'interès en formati document de text.
  - Imatge on es dibuixen els punts d'interès trobats. El format d'imatge de sortida és PNG.
  - Document de text on és s'imprimeixen la llista de coincidències que hi ha entre les imatges. En el moment en què les imatges es relacionen entre elles, s'ha definit en el codi, que una imatge només es relaciona amb les 3 imatges que van just davant d'ella. És a dir, si introduïm 5 imatges, la relació entre elles serà la següent.

Imatge 1	Imatge 2	Imatge 3	Imatge 4	Imatge 5
x	x			
x		x		
x			x	
	x	x		
	x		x	
	x			x
		x	x	
		x		x
			x	x

Taula 2.3: Explicació de relació entre imatges

Fent que per exemple en aquest cas on s'han introduït 5 imatges hi hagi un total de 9 relacions. Això és important a tenir en compte a l'hora d'utilitzar un bloc d'imatges molt gran, ja que la memòria ja està reservada per a realitzar aquesta operació, i si és sobrepasses el programa deixaria de funcionar.

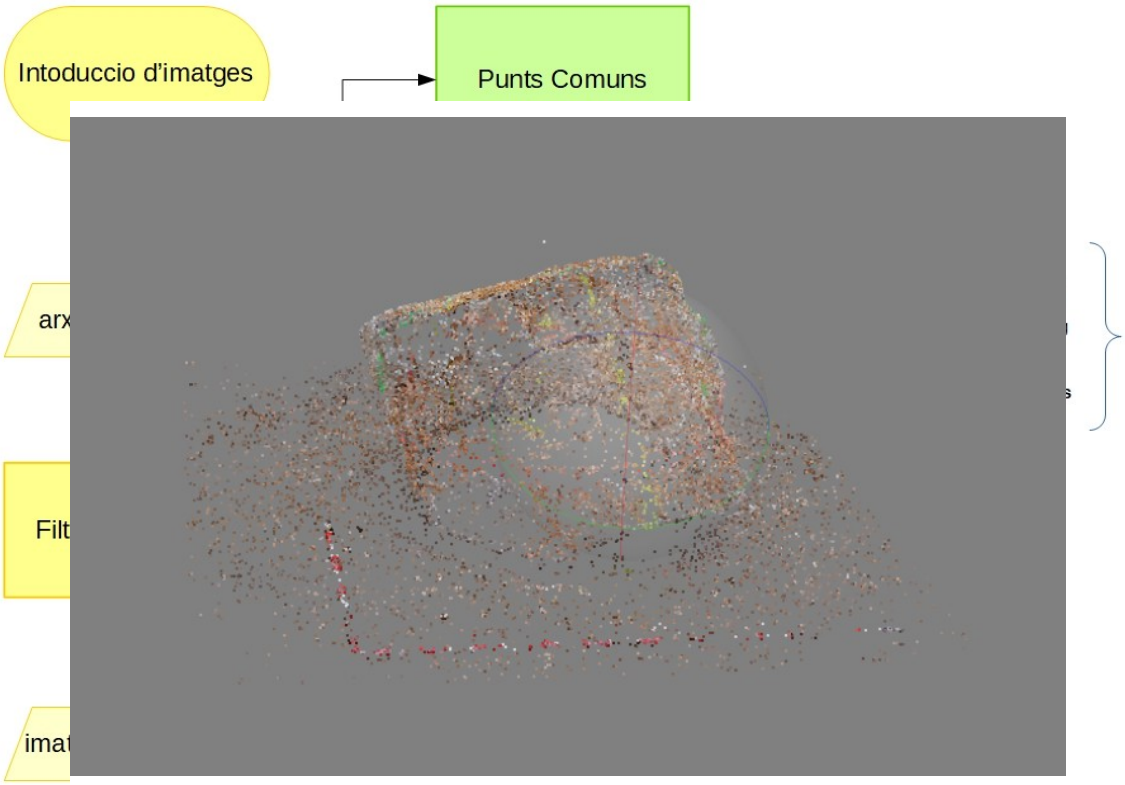
  - Imatge on podem observar les coincidències, representades en línies, entre dues imatges.
  - Document de text amb la matriu d'homografia de dues imatges.
  - Document de text amb la matriu fonamental de dues imatges.
  - Coordenades píxel d'un llistat de punts, en format de document de text. Aquest és el document necessari que s'haurà d'introduir en el mòdul 4. A aquest llistat de punts s'ha refinat el càlcul de l'error de la matriu fonamental, per a eliminar aquells punts que tenen un error molt groller.

Posteriorment, els documents resultants (dues llistes de punts per cada procés de coincidència), s'han de fusionar en un de sol. Aquest document de text resultant és una matriu molt gran, on en aquells llocs on no hi han coincidències s'haurà de donar-li un valor de -1.

2743	1707	3076	1457	3067	1366	-1	-1	-1	-1	-1	-1
3700	549	2827	415	1913	466	-1	-1	-1	-1	-1	-1
2696	800	1562	772	208	922	-1	-1	-1	-1	-1	-1
3692	576	2824	440	1912	492	-1	-1	-1	-1	-1	-1
2868	982	3001	770	2858	731	-1	-1	-1	-1	-1	-1
2868	982	3001	770	2858	731	-1	-1	-1	-1	-1	-1
2868	982	3001	770	2858	731	-1	-1	-1	-1	-1	-1
3638	553	2764	422	1844	475	-1	-1	-1	-1	-1	-1
3047	1687	3338	1395	3253	1290	-1	-1	-1	-1	-1	-1
3621	757	2766	617	1859	675	-1	-1	-1	-1	-1	-1
2764	1032	2904	828	2781	790	-1	-1	-1	-1	-1	-1
2464	1125	2676	941	2642	904	-1	-1	-1	-1	-1	-1
1858	1696	2268	1577	2482	1542	-1	-1	-1	-1	-1	-1
2766	1537	3063	1295	3030	1216	-1	-1	-1	-1	-1	-1
-1	-1	2082	1842	2316	1837	2594	1859	-1	-1	-1	-1
-1	-1	2076	1787	2306	1782	2578	1799	-1	-1	-1	-1
-1	-1	4960	557	3722	566	2927	466	-1	-1	-1	-1
-1	-1	2424	861	2425	833	2461	748	-1	-1	-1	-1
-1	-1	4950	551	3714	562	2918	462	2253	520	1736	593
-1	-1	2062	1704	2284	1698	2550	1710	-1	-1	-1	-1
-1	-1	2060	1727	2283	1723	2551	1737	-1	-1	-1	-1
-1	-1	2441	884	2444	856	2482	772	-1	-1	-1	-1
-1	-1	2052	1709	2274	1705	2540	1718	-1	-1	-1	-1
-1	-1	4042	651	2936	677	2120	616	1405	716	793	820
-1	-1	2716	946	2676	908	2677	818	-1	-1	-1	-1
-1	-1	2460	2118	2710	2057	2978	2036	-1	-1	-1	-1
-1	-1	5338	843	4120	780	3392	652	-1	-1	-1	-1
-1	-1	4927	641	3710	637	2927	536	-1	-1	-1	-1
-1	-1	2469	818	2459	790	2484	699	-1	-1	-1	-1
-1	-1	2038	1476	2252	1464	2506	1454	-1	-1	-1	-1
-1	-1	5352	692	4046	666	3256	550	-1	-1	-1	-1
-1	-1	-1	-1	2982	624	2159	558	1437	653	822	750
-1	-1	-1	-1	2982	624	2159	558	1437	653	822	750
-1	-1	-1	-1	3299	535	2492	451	1800	527	-1	-1
-1	-1	-1	-1	1650	2607	1881	2849	2456	3031	3119	3175
-1	-1	-1	-1	3303	1899	3443	1792	3617	1735	-1	-1
-1	-1	-1	-1	2963	680	2149	617	1436	716	829	819
-1	-1	-1	-1	1616	1266	1774	1299	2317	1382	-1	-1
-1	-1	-1	-1	4008	212	2801	142	1757	221	-1	-1
-1	-1	-1	-1	4620	1053	3879	883	3293	876	-1	-1
-1	-1	-1	-1	3652	566	2854	468	2185	528	1659	603
-1	-1	-1	-1	2264	1484	2521	1475	2988	1499	-1	-1
-1	-1	-1	-1	3556	581	2759	487	2087	552	1554	629
-1	-1	-1	-1	3666	553	2864	456	2192	515	1663	589
-1	-1	-1	-1	3672	574	2875	475	2208	534	-1	-1
-1	-1	-1	-1	3979	186	2771	118	1725	197	-1	-1

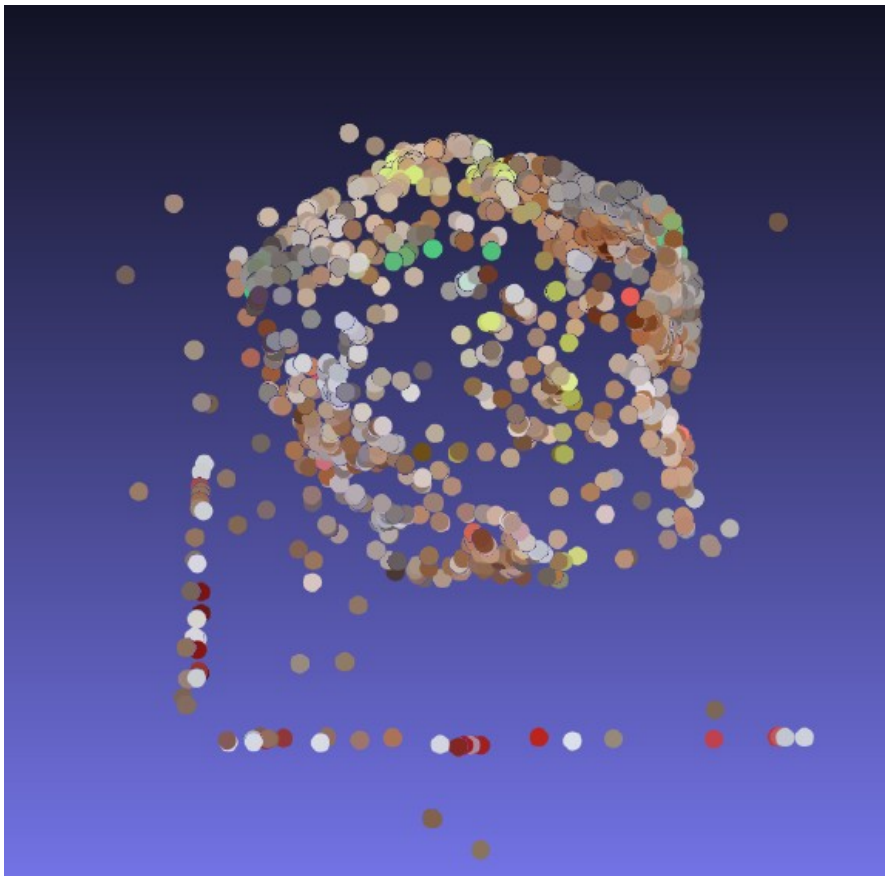
Imatge 2.32: Exemple de document de coordenades 2d ja fusionat

Finalment tindríem l'últim mòdul és de generació 3d. Aquest mòdul genera a partir de la funció *cv::sfm::reconstruct()* de la llibreria OpenCV, la construcció d'un document de text on apareixen una llista dels nostres punts en coordenades x, y i z. Amb aquesta llista de punts i amb l'ajuda d'un programa extern de visualització de models digitals, podem observar el resultat final.



Imatge 2.36: Resultats del model digital obtinguts amb el programa Agisoft Photoscan

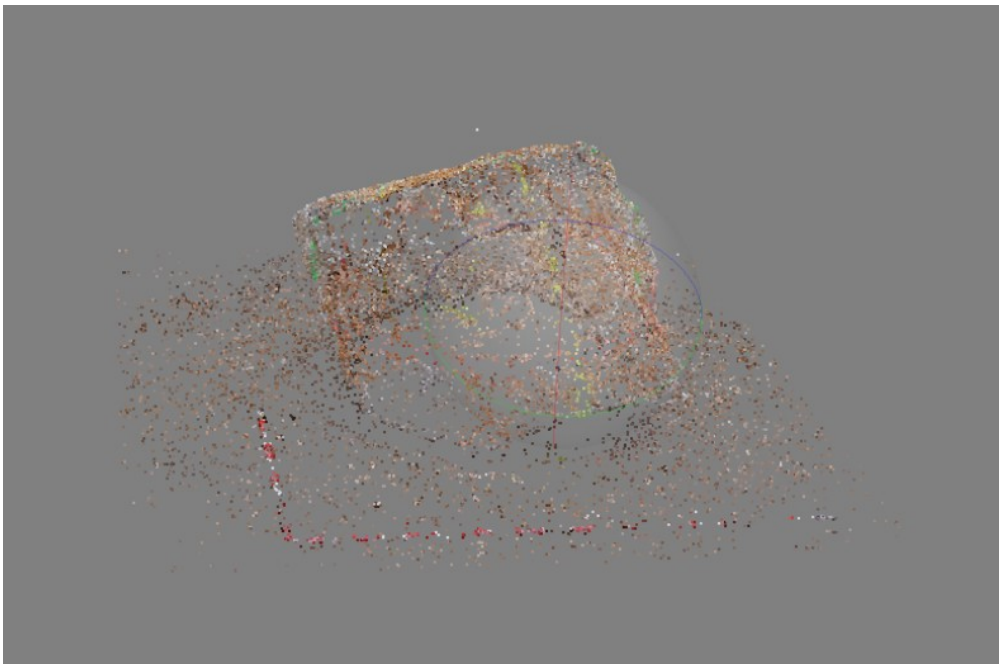
Un cop s'han creat tots els mòduls, s'ha volgut realitzar un model digital. Per a la realització d'aquest model, s'han escollit unes tomes fotogràfiques d'una roca. El resultat final de l'execució de tots els mòduls ens ha generat un model on podem diferenciar perfectament la roca.



Imatge 2.34: Model resultant del nostre programa

Imatge 2.35: Imatge d'origen d'informació

Aquest model també s'ha creat paral·lelament amb un programa comercials de creació de models digitals, on podem observar uns resultats semblants als obtinguts, però amb molts més punts trobats.



Imatge 2.36: Resultats del model digital obtinguts amb el programa Agisoft Photoscan

3 CONCLUSIONS

Els resultats finals obtinguts, després de la realització de tots els mòduls, ha estat el que es buscava. Primerament, i l'objectiu més clar del treball, és que s'ha obtingut el model digital que es buscava des d'un principi, i el resultat ha estat el que s'espera. Per una altra banda, s'ha complert amb l'objectiu que el programa funciona amb un bloc fotogràfic gran, fent així que no es hagi de realitzar el procés imatge per imatge, sinó que el programa pot carregar tot el bloc fotogràfic de cop. En el nostre cas hem fet proves amb un bloc de 42 imatges.

La implementació del programa s'ha fet en llenguatge C++ corrent sobre el sistema operatiu Windows 10, cosa que es podria haver fet de manera diferent; per exemple en Linux.

El C++ és un llenguatge de programació molt potent; però es podria haver arribat al mateix resultat amb la utilització de Python. Python és un llenguatge de programació, no tant potent com C++, però si molt més senzill a la pràctica, per aquest motiu molts dels fòrums i exemples que es troben per la xarxa estan orientats a treballar amb Python.

El problema que tenim amb Windows és la instal·lació d'algunes llibreries, sobretot el problema ha aparegut alhora de la instal·lació del mòdul extern de SFM on s'havia de pre-compilar algunes llibreries externes (Eigen, Glog, Gflags i Ceres Solver). Si s'hagués utilitzat Ubuntu (un sistema operatiu particular de Linux), la instal·lació d'aquestes llibreries s'hauria reduït a unes poques línies de codi i de temps emprat.

També voldria mencionar que el temps emprat des del primer pas fins a obtenir resultats és força llarg en comparació al software comercial que podem trobar; però com que els resultats són força bons fer servir aquest projecte és una bona opció. És evident que en aquest treball només es volia traçar el fil conductor per passar de fotografies a un model tridimensional i, ara quedaria un llarg procés d'optimització de les rutines.

4 BIBLIOGRAFIA

Bay, H [a]. Ess, A[a]. Tuytelaars, T [b]. i Van Gool, L [ab]. (2008), Speeded-Up Robust Feature (SURF). [a] ETH Zurich, Zuric, Suïssa. [b] K. U. Leuven, ESAT-PSI, Leuven, Bèlgica.

Calonder, M. Lepetit, V. Strecha, C. i Fua, P. BRIEF: Binary Robust Independent Elementary Features. CVLab, EPFL, Lausanne, Suïssa.

Hartley, R. i Zisserman, A. (2004), Multiple View Geometry in computer vision. Cambridge University Press, Cambridge, Regne Unit .

Kaehler, A. i Bradski, G. (2017), Learning OpenCV 3 Computer Vision in C+ with the OpenCV library. O'Reilly Media, Inc., Sebastopol, Ucraïna.

Lowe, D.G. (2004), Distintive Image Features from Scale-Invariant Keypoints. Computer Science Departament, University of British Columbia, Vancouver, B.C., Canada.

Panchal, P.M[a]. Panchal, S.R [b]. Shah, S.K [c]. (2013). A Comparison of SIFT and SURF. [a]PG Student, Department of Electronics & Communication Engineering, SVIT, Vasad, India. [b]Research Scholar, Department of Electronics & Communication Engineering, CHARUSAT, Changa, India. [c]Professor, Department Electrical Engg., Faculty of Tech. & Engg., M S University of Baroda, Vadodara, India. Dsiponible a <https://pdfs.semanticscholar.org/1823/525af5a9ee0790f70b40de5e783c8ded9aa8.pdf>.

Rublee, E. Rabaud, V. Konolige, K. i Bradski, G. ORB: an efficient alternative to SIFT or SURF. Willow Garage, Menlo Park, California, Estats Units d'Amèrica.

DOCUMENTACIÓ DIGITAL

Akalanska Perera, S. (2018). A comparison of SIFT, SURF and ORB. Disponible a <https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea>.

fdxLABS. (2019). Calculate X, Y, Z Real World Coordinates from Image Coordinates using OpenCV. Disponible a <https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/>.

Mallick, S. (2018). Image Alignment (Feature Based) using OpenCV (C++/Python). Disponible a <https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/>.

Mallick, S. (2016). Homography Examples using OpenCV (Python / C++). Disponible a <https://www.learnopencv.com/homography-examples-using-opencv-python-c/>.

OpenCV. OpenCV 2.4.13.7 documentation. Disponible a <https://docs.opencv.org/2.4/modules/refman.html>.

OpenCV. (2019). The OpenCV Tutorials Release 2.4.13.7. Disponible a <https://docs.opencv.org/2.4/opencv_tutorials.pdf>.

OpenCV. Tutorials for contrib modules. Disponible a <https://docs.opencv.org/master/de/d7c/tutorial_table_of_content_sfm.html>.

OpenCV. OpenCV Tutorials. Disponible a <https://docs.opencv.org/3.4/d9/df8/tutorial_root.html>.

Robotics with Ros. Finding Homography Matrix using Singular-value Decomposition and RANSAC in OpenCV and Matlab. Disponible a <http://ros-developer.com/2017/12/26/finding-homography-matrix-using-singular-value-decomposition-and-ransac-in-opencv-and-matlab/>.

Singh, C.D. Structure from Motion. University of Maryland CS. College Park, Maryland, Estats Units. Disponible a <https://cmsc426.github.io/sfm/>.

Universitat de Winsconsin, Nevada. The Gaussian kernel. Disponible a <http://pages.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf>.

Weitz, E. (2016). SIFT - Scale-Invariant Feature Transform. Disponible a <http://weitz.de/sift/ >.

5 ANNEXOS

5.1. Programa de lectura de fitxers

```
// IntroduccioImatges.cpp : Defines the entry point for the console
application.
//
#include "stdafx.h"
 #include <stdio.h>
 #include <sys/types.h>
 #include "dirent.h"
 #include <iostream>
 #include <fstream>
 #include <stdlib.h>

 using namespace std;

     int main (void)
     {
       DIR *dp;
        FILE *dc; //faig nou fitxer on es guardara la llista de arxius que
hi han a la carpeta seleccionada
       struct dirent *ep;
       char folder[100];
       char* j;
       ifstream inn;
              const  char  *dirname="C:\\Users\\David\\Desktop\\esborrar\\
ImatgesModel"; //S'ha de ficar doble barra a la ruta
       dp = opendir (dirname);
       dc = fopen("arxiusCarpeta.txt", "wt");
       printf("Obrint carpeta %s: \n", dirname);
       if (dp != NULL)
         {
           while (ep = readdir (dp))
           {
             j=ep->d_name;
             sprintf(folder,"C:/folder/%s",j);
             fputs(dirname, dc);
             fputs("\\",dc);
             fputs(j, dc);
             fputs("\n", dc);
             cout<<folder<<endl;
           }
           (void) closedir (dp);
           fclose (dc);
         }
       else
       {
         perror ("No es pot obrir la carpeta especificada");
       }

       system("pause");
       return 0;
     }
```

5.2. Programa de filtratge d'imatges

```
// filtratgeImatges.cpp : Defines the entry point for the console
application.
//
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>



int _tmain(int argc, _TCHAR* argv[])
{
    FILE *fp;
    FILE *dc;
    char fi[5][10] = {".jpg",".jpeg",".png",".tif",".tiff"}; //Aquí guardo
tots els formats admesos en el programa
    char temp[1024]; //Array temporal on es guardaran les línies del
document arxiusCarpeta.txt
    int j=0;
    const char punt = '.';
    char *ret;
    char *retm;
    int comp;

    char t1[6]={".jpg"};
    char t2[5]={".jpg"};
    comp= strcmp(t1,t2);  // Dues cadenes exactes fa que la funció retorni
0

    char t3[6]={".jpg\r"};
    char t4[5]={".jpg"};
    comp= strcmp(t3,t4); // Dues cadenes quasi exactes fa que la funció
retorni 1. En aquest cas t3 té un "carriage return" novisible en ASCII.

    fp=fopen("arxiusCarpeta.txt", "r"); //Obro arxiu anterior
    dc=fopen("imatgesFiltrades.txt","wt"); //Creo arxiu resultat


    while(fgets(temp, 1024, (FILE*) fp)) //Amb el while anem passant línia
per línia del fitxer
        {
            /// Solució: agafem l'últim char de la cadena (que conté el \r
o CR) i el canviem per un zero (final de cadena).
            int longitud= strlen(temp);
            temp[longitud-1]='\x0'; // final de cadena
            ////////////////////////////////////////////////////////////
            printf("%s\n", temp);
            ret = strrchr(temp, punt); //Ens talla per la dreta fins al
punt -> .(tipus de document)
            retm = strlwr(ret); //Passem a minúscula
            for(j=0;j<=4;j++){ //Fem el for per a moure'ns a traves del
array fi

            //-------------------------------------------------------------
                comp = strcmp(retm, fi[j]);
```

```
                printf("Cas %d: %s i %s dona %d\n",j, retm, fi[j], comp);
                if(comp==0){
                    fputs(temp,dc);
                    fputs("\n",dc);
                }
            }
            printf("\n----------------------\n");
        }

    fclose(fp);
    fclose(dc);
    system("pause");
    return 0;
}
```

5.3. Programa de cerca de punts comuns

```
// puntComu.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include "opencv2/core.hpp"
#include <fstream>
#include <iostream>
#include <string>
#include <stdio.h>
#include <vector>



#ifdef HAVE_OPENCV_XFEATURES2D
#include "opencv2/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"
#include "opencv2/xfeatures2d/nonfree.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2\stitching\detail\autocalib.hpp"



using namespace cv;
using namespace cv::xfeatures2d;
using namespace std;


int _tmain(int argc, _TCHAR* argv[])
{
    FILE *fp;

    char temp[1024];
    char temp2[1024];
    int i=0;

    fp=fopen("imatgesFiltrades.txt","r");

    if (fp==NULL){
        exit(0);
    }


    Mat img_array[42]; //capat a 10 imatges*/
    Mat img_red[42]; //aray de les imatges reduides
    int e=98;

    ////////////////////////////////////////////////////////////////
    //En aquest while guardem dintre del nostre array les imatges//
    ////////////////////////////////////////////////////////////////

puts("// LECTURA D'IMATGES:");
    while(fgets(temp, 1024, (FILE*) fp))
    {
```

```
        int longitud= strlen(temp);
        temp[longitud-1]='\x0';
        printf("    Llegint imatge numero %d",i+1);
        img_array[i]=imread(temp,CV_LOAD_IMAGE_COLOR);
        printf(" ------ Introduida\n");
        i=i+1;
    }

    fclose(fp);
    int t=i;

puts("// REDUCCIO TAMANY IMATGE:");
    for (i=0;i<t;i++)
    {
        printf("    Reduint imatge numero %d",i+1);
        resize(img_array[i], img_red[i], Size(), 1.0/ 2.0, 1.0/ 2.0,
CV_INTER_CUBIC);
        printf(" ------ Reduida\n");
    }

//-- Step 1: Detect the keypoints:
    Mat img_keypoints[42];
    Mat descriptors[42];
    std::vector<KeyPoint> keypoints[42];

puts("\n    Crea objecte SIFT");
    cv::Ptr<xfeatures2d::SIFT> f2d = xfeatures2d::SIFT::create(1000);
//Aqui podem posar quant punts volem que en trobi a cada imatge

    ////////////////////////////////////////
    //Noms i extensions de fitxers sortints//
    ////////////////////////////////////////

    char n[]= "Keypoints"; //Necesito aquest char per a després em faciliti
el posar-hi nom als txt
    char a[]= ".txt";
    char ni[]= "ImatgeKeypoints";
    char ai[]= ".png";
    char nm[]= "Matches_";
    char nc[]= "MatchesCorners_";
    char se[]= "_";
    char ho[]= "Homografia_";
    char mo[]= "Fonamental_";
    char pt[]= "_puntsImatge_";

puts("// DETECCIO DE KEYPOINTS:\n");
    for (i=0;i<t;i++){

//Detecció de key points
    printf("Detecta keypoints d'imatge %d", i+1);
    cv::setBreakOnError(true);
    f2d->detect(img_red[i], keypoints[i] );
    printf(" ------ Finalitzat\n");

//Quantitat de keypoints trobats
    int NumPoints= (int)keypoints[i].size();
    printf("Detectats %d punts en imatge %d\n", NumPoints, i+1 );
```

```
//Impresió de keypoints en txt
puts("Imprimint keypoints a .txt");

        sprintf(temp, "%s%d%s", n, i+1, a);
        FileStorage fs(temp, FileStorage::WRITE);
        write( fs , "Keypoint", keypoints[i]);
        fs.release();

//Impresió de img_keypoints en jpg
puts("Imprimint imatges amb keypoints a .png");
        drawKeypoints( img_red[i], keypoints[i], img_keypoints[i] );
        sprintf(temp, "%s%d%s", ni, i+1, ai);
        imwrite(temp, img_keypoints[i]);

//-- Step 2: Calculate descriptors (feature vectors)
    printf("Calculant descriptors");
    f2d->compute(img_red[i], keypoints[i], descriptors[i]);
    printf(" ------ Finalitzat\n");

        printf("\n      -------------\n\n");
    }

//-- Step 3: Matching descriptor vectors using FLANN matcher

    //Per a saber quants matxes em de buscar

    int s=3; //Intervals de comparació d'imatge
    int m; //Matches
    int sum=0; //Sumatori
    for (i=1; s>=i; i++)
    {
        sum += s-i;
    }
    m=(t-s)*s+sum; //Això ens calcula la quantitats de matches que s'han de
crear
    puts("\nGeneracio de matches");
    printf("\nTotal d'imatges -> %d\nIntervals entre imatges -> %d\nNumero
de matches -> %d\n\n", t, s, m);

    //BFMatcher matcher;
    std::vector< DMatch > matches;
    std::vector< DMatch > good_matches;
    Mat img_matches[200];
    //BFMatcher matcher;
    BFMatcher matcher(NORM_L2, true);
    int k, j, y;
    j=0;
    k=t-1;

    for(i=0; i<=k; i++){
        for(y=i+1; y!=(i+s+1); y++){
            if(y<t){
                Mat descriptor_object;
                Mat descriptor_scene;

                descriptors[i].copyTo(descriptor_object);
                descriptors[y].copyTo(descriptor_scene);
```

```
                printf("Creant matches numero %d/%d. Imatge %d -> Imatge
%d\n", j+1, m, i+1, y+1);
                matches.clear();
                good_matches.clear();
                matcher.match( descriptor_object, descriptor_scene,
matches);

                printf("    Depurant punts...\n");

                double max_dist = 0;
                double min_dist = 100;

                for( int w = 0; w < matches.size(); w++ )
                {
                    double dist = matches[w].distance;
                    if( dist < min_dist ) min_dist = dist;
                    if( dist > max_dist ) max_dist = dist;
                }
                printf("    -- Max dist : %f \n", max_dist );
                printf("    -- Min dist : %f \n", min_dist );

                for( int w = 0; w < matches.size(); w++ )
                {
                    if( matches[w].distance <= 3*min_dist )
                    {
                        good_matches.push_back( matches[w]);
                    }
                }

                //Impresió de goodmatches en txt
                printf("Imprimint goodmatches a .txt\n");

                sprintf(temp, "%s%d%s%d%s", nm, i+1, se, y+1, a);
                FileStorage fx(temp, FileStorage::WRITE);
                write( fx , "Goodmatches", good_matches);
                fx.release();


                //Dibuix i impresio de img_matches goodmatches
                printf("Dibuixant imatges goodmatches \n");
                drawMatches(img_red[i], keypoints[i], img_red[y],
keypoints[y], good_matches, img_matches[j], Scalar::all(-1), Scalar::all(-
1),
                std::vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
                printf("Imprimint imatges goodmatches \n");
                sprintf(temp, "%s%d%s%d%s", nm, i+1, se, y+1, ai);
                imwrite(temp, img_matches[j]);


                //Detecció d'objectes
                std::vector<Point2f> obj;
                std::vector<Point2f> scene;
                std::vector<Point2f> obj2;
                std::vector<Point2f> scene2;

                std::vector<KeyPoint> keypoints_object;
                std::vector<KeyPoint> keypoints_scene;
```

```
                obj.clear();
                scene.clear();
                obj2.clear();
                scene2.clear();
                keypoints_object.clear();
                keypoints_scene.clear();

                //Obrir .txt keypoints_object & keypoints_scene
                sprintf(temp, "%s%d%s", n, i+1, a);
                FileStorage fo(temp, FileStorage::READ);
                read(fo["Keypoint"], keypoints_object);
                fo.release();

                sprintf(temp, "%s%d%s", n, y+1, a);
                FileStorage fs(temp, FileStorage::READ);
                read(fs["Keypoint"], keypoints_scene);
                fs.release();


                for( size_t w = 0; w < good_matches.size(); w++ )
                {
                    //-- Get the keypoints from the good matches

    obj.push_back( keypoints_object[ good_matches[w].queryIdx ].pt );

    scene.push_back( keypoints_scene[ good_matches[w].trainIdx ].pt );
                }



                puts("Creant homografia");
                Mat H;
                H.release();
                H = findHomography( obj, scene, RANSAC );
                cout << "Homografia=" << endl<< H << endl << endl;
                puts("Creant matriu fonamental");
                Mat F;
                F.release();
                F = cv::findFundamentalMat(obj, scene, RANSAC, 3.0, 0.999
);
                cout << "Fonamental=" << endl<< F << endl << endl;

                sprintf(temp, "%s%d%s%d%s", ho, i+1, se, y+1, a);
                FileStorage fh(temp, FileStorage::WRITE);
                write( fh , "Homografia", H);
                fh.release();

                sprintf(temp, "%s%d%s%d%s", mo, i+1, se, y+1, a);
                FileStorage fn(temp, FileStorage::WRITE);
                write( fn , "Fonamental", F);
                fn.release();

                // Això va just després del càlcul de la matriu fonamental
                // vector v1 (u1,v1,1), v2(u2,v2,1) (coordenades imatge (en
píxels))
                Mat v1(1, 3, CV_64FC1), v2(3, 1, CV_64FC1);
```

```
                // derivades del producte S= vec1t * F * vec2 respecte u1,
v1, u2 i v2
                Mat dv1dx(1, 3, CV_64FC1), dv1dy(1, 3, CV_64FC1), dv2dx(3,
1, CV_64FC1), dv2dy(3, 1, CV_64FC1);

                //correctMatches(F, obj, scene, obj2, scene2); // Això no!!

                double res;
                double h = 0.1;
                vector<Point2f>::iterator it2 = obj.begin();
                vector<KeyPoint>::iterator itkeyobj =
keypoints_object.begin();

                vector<KeyPoint>::iterator itkeysce =
keypoints_scene.begin();

                for (vector<Point2f>::iterator it = scene.begin(); it !=
scene.end(); it++, it2++, itkeyobj++ , itkeysce++ )
                {
                    double x1, y1, z1;
                    // carreguem v1 i v2 amb les dades del punt vist des de
obj i des de scene
                    v1.at<double>(0, 0) = it->x;
                    v1.at<double>(0, 1) = it->y;
                    v1.at<double>(0, 2) = 1.0;
                    v2.at<double>(0, 0) = it2->x;
                    v2.at<double>(1, 0) = it2->y;
                    v2.at<double>(2, 0) = 1.0;

                    // preparem la derivada parcial dS/du1  "les "d" són
cursives de derivada parcial"
                    dv1dx.at<double>(0, 0) = it->x + h;
                    dv1dx.at<double>(0, 1) = it->y;
                    dv1dx.at<double>(0, 2) = 1.0;

                    // preparem la derivada parcial dS/dv1  "les "d" són
cursives de derivada parcial"
                    dv1dy.at<double>(0, 0) = it->x;
                    dv1dy.at<double>(0, 1) = it->y + h;
                    dv1dy.at<double>(0, 2) = 1.0;

                    // preparem la derivada parcial dS/du2  "les "d" són
cursives de derivada parcial"
                    dv2dx.at<double>(0, 0) = it2->x + h;
                    dv2dx.at<double>(1, 0) = it2->y;
                    dv2dx.at<double>(2, 0) = 1.0;

                    // preparem la derivada parcial dS/dv2  "les "d" són
cursives de derivada parcial"
                    dv2dy.at<double>(0, 0) = it2->x;
                    dv2dy.at<double>(1, 0) = it2->y + h;
                    dv2dy.at<double>(2, 0) = 1.0;

                    //Càlcul de les S = vec1t * F * vec2
                    Mat valor = v1*F*v2;
                    Mat valordx = dv1dx*F*v2;
                    Mat valordy = dv1dy*F*v2;
                    Mat valordx2 = v1*F*dv2dx;
```

```
                    Mat valordy2 = v1*F*dv2dy;

                    // Càlcul del diferencial
                    //dS = dS/du1 du1 + dS/dv1 dv1 + dS/du1 du2 + dS/dv2
dv2
                    double der = fabs(valor.at<double>(0, 0) -
valordx.at<double>(0, 0)) / h + fabs(valor.at<double>(0, 0) -
valordy.at<double>(0, 0)) / h + fabs(valor.at<double>(0, 0) -
valordx2.at<double>(0, 0)) / h + fabs(valor.at<double>(0, 0) -
valordy2.at<double>(0, 0)) / h;

                    //Càlcul de  vec1t * F * vec2
                    x1 = F.at<double>(0, 0)*it2->x + F.at<double>(0,
1)*it2->y + F.at<double>(0, 2);
                    y1 = F.at<double>(1, 0)*it2->x + F.at<double>(1,
1)*it2->y + F.at<double>(1, 2);
                    z1 = F.at<double>(2, 0)*it2->x + F.at<double>(2,
1)*it2->y + F.at<double>(2, 2);
                    res = x1*it->x + y1*it->y + z1;

                    if (fabs(res) <= 3.0*fabs(der)) // si el producte S
supera en 3 vegades el diferencial estimat per a variacions d'un píxel
nol'aceptis
                    {
                        scene2.push_back(*it);
                        obj2.push_back(*it2);
                    }
                    //else
                    //  cout << "eliminate point " <<  res << " " << der <<
" " << res / der << endl;
                }


                cout << "IMATGE" << i << endl << "PUNTS PRIMERA VOLTA ->"
<< obj.size() << endl << "PUNTS SEGONA VOLTA ->" << obj2.size() << endl;
                cout << "IMATGE" << y << endl << "PUNTS PRIMERA VOLTA ->"
<< scene.size() << endl << "PUNTS SEGONA VOLTA ->" << scene2.size() <<
endl;

                // comprovació de que la matriu fonamental no ha canviat
massa després del filtratge
                Mat FF;
                FF.release();
                FF = cv::findFundamentalMat(obj2, scene2, RANSAC, 3.0,
0.999);

                cout << "Fonamental post=" << endl << FF << endl << endl;
                //              system("PAUSE");
                //Impressio de punts a format txt
                puts("Impressió de punts");

                sprintf(temp, "%d%d%s%d%s", i + 1, y + 1, pt, i + 1, a);
                FileStorage fl(temp, FileStorage::WRITE);
                write(fl, "Coordenades", obj2);
                fl.release();

                sprintf(temp, "%d%d%s%d%s", i + 1, y + 1, pt, y + 1, a);
                FileStorage fb(temp, FileStorage::WRITE);
```

```
                write(fb, "Coordenades", scene2);
                fb.release();


                sprintf(temp, "%sBons_%d%s%d%s", nm, i + 1, se, y + 1, a);
                FILE *output;
                fopen_s(&output, temp, "w");

                if ((double)obj2.size() / (double)obj.size() > 0.5)
                {
                    vector<Point2f>::iterator punt2 = scene2.begin();
                    fprintf_s(output, "%d\n", (int)scene2.size());
                    for (vector<Point2f>::iterator punt = obj2.begin();
punt != obj2.end(); punt++, punt2++)
                        fprintf_s(output, "%12.6lf %12.6lf  %12.6lf
%12.6lf\n", punt->x, punt->y, punt2->x, punt2->y);

                    fclose(output);
                }
                printf("--------\n\n");

                j=j+1;
            }
        }
    }

    waitKey(0);
    system("pause");

    return 0;
}

#endif
```

5.4. Filtratge de punts a partir de la matriu d'homografia

```
// img2.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include "opencv2/core.hpp"
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <array>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#ifdef HAVE_OPENCV_XFEATURES2D
#include "opencv2/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"
#include "opencv2/xfeatures2d/nonfree.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/core/types.hpp"
#include "opencv2/stitching/detail/autocalib.hpp"
#include "opencv2\core\types.hpp"

using namespace cv;
using namespace cv::xfeatures2d;
using namespace std;


int _tmain(int argc, _TCHAR* argv[])
{

puts("Obrint imatges...");
    Mat img1=imread("C:\\Users\\David\\Desktop\\esborrar\\img2\\img2\\
DibuixScan\\conjunt.JPG");
    Mat img2=imread("C:\\Users\\David\\Desktop\\esborrar\\img2\\img2\\
DibuixScan\\db3.JPG");
    Mat img1_r;
    resize(img1, img1_r, Size(), 1.0/ 2.0, 1.0/ 2.0, CV_INTER_CUBIC);
    Mat img2_r;
    resize(img2, img2_r, Size(), 1.0/ 2.0, 1.0/ 2.0, CV_INTER_CUBIC);

std::puts("Detectant punts d'interes...");
    Mat img_kp1, img_kp2;
    Mat dt1, dt2;
    std::vector<KeyPoint> kp1, kp2;

    cv::Ptr<xfeatures2d::SIFT> f2d = xfeatures2d::SIFT::create(1000);

    cv::setBreakOnError(true);
    f2d->detect(img1_r, kp1);
    f2d->detect(img2_r, kp2);

    puts("Imprimint llistat de punts d'interes e imatges...");
```

```
    FileStorage fs("Keypoints1.txt", FileStorage::WRITE);
    write( fs , "Keypoint", kp1);
    fs.release();

    FileStorage fp("Keypoints2.txt", FileStorage::WRITE);
    write( fp , "Keypoint", kp2);
    fp.release();

    drawKeypoints( img1_r, kp1, img_kp1, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("Keypoints1.png", img_kp1);
    drawKeypoints( img2_r, kp2, img_kp2, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("Keypoints2.png", img_kp2);

    f2d->compute(img1_r, kp1, dt1);
    f2d->compute(img2_r, kp2, dt2);


    puts("Creant matches...");
    Mat img_matches;
    Mat img_matches_nf;
    std::vector< DMatch > matches;
    std::vector< DMatch > good_matches;
    BFMatcher matcher(NORM_L2, true);
    matcher.match( dt1, dt2, matches );

    cout << "Quantitat de matcehs -> " << matches.size() << endl;
    double max_dist = 0;
    double min_dist = 100;

    for( int w = 0; w < matches.size(); w++ )
    {
        double dist = matches[w].distance;
        if( dist < min_dist ) min_dist = dist;
        if( dist > max_dist ) max_dist = dist;
    }
    printf("    -- Max dist : %f \n", max_dist );
    printf("    -- Min dist : %f \n", min_dist );

    for( int w = 0; w < matches.size(); w++ )
    {
        if( matches[w].distance <= 3*min_dist )
        {
            good_matches.push_back( matches[w]);
        }
    }

    puts("Creant imatges amb matches...");
    drawMatches(img1_r, kp1, img2_r, kp2, matches, img_matches_nf,
Scalar::all(-1), Scalar::all(-1), std::vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    imwrite("Matches_nf.png", img_matches_nf);
    drawMatches(img1_r, kp1, img2_r, kp2, good_matches, img_matches,
Scalar::all(-1), Scalar::all(-1), std::vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    imwrite("Matches.png", img_matches);
```

```
    FileStorage fx("Matches.txt", FileStorage::WRITE);
    write( fx , "Matches", good_matches);
    fx.release();

    printf("\nNumero de keypoints d'imatge 1 ---> %d\nNumero de keypoints
d'imatge 2 ---> %d\nQuantitat de matches ---> %d\n\n", kp1.size(),
kp2.size(), good_matches.size());

    std::vector<Point2f> obj;
    std::vector<Point2f> scene;

    for( size_t i=0 ; i <matches.size() ; i++ )
    {
        obj.push_back( kp1[ matches[i].queryIdx ].pt );
        scene.push_back( kp2[ matches[i].trainIdx ].pt );
    }

    FileStorage fo("Obj.txt", FileStorage::WRITE);
    write( fo , "Obj", obj);
    fo.release();

    FileStorage fc("Scene.txt", FileStorage::WRITE);
    write( fc , "Scene", scene);
    fc.release();

    Mat H= findHomography( obj, scene, RANSAC, 3.0 );

    std::cout << "Homografia = " << endl << " "  << H << std::endl <<
std::endl;

    Mat F= findFundamentalMat( obj, scene, RANSAC, 3.0, 0.999 );

    FileStorage fr("Homografia.txt", FileStorage::WRITE);
    write( fr , "Homografia", H);
    fr.release();

    FileStorage fy("Fonamental.txt", FileStorage::WRITE);
    write( fy , "Fonamental", F);
    fy.release();

    double h11 = H.at<double>(0,0);
    double h12 = H.at<double>(0,1);
    double h13 = H.at<double>(0,2);
    double h21 = H.at<double>(1,0);
    double h22 = H.at<double>(1,1);
    double h23 = H.at<double>(1,2);
    double h31 = H.at<double>(2,0);
    double h32 = H.at<double>(2,1);
    double h33 = H.at<double>(2,2);

    puts("Posicio de píxel en Obj & Scene");
    std::vector<Point2f> obj2;
    std::vector<Point2f> scene2;
    int valor_min=1;
    puts("Calcul d'error");
    for (int i=0; i<obj.size(); i++)
    {
```

```
        double err_x = ((scene[i].x)-((h11*obj[i].x+h12*obj[i].y+h13)/
(h31*obj[i].x+h32*obj[i].y+h33)));
        double err_y = ((scene[i].y)-((h21*obj[i].x+h22*obj[i].y+h23)/
(h31*obj[i].x+h32*obj[i].y+h33)));
        double err= (err_x*err_x)+(err_y*err_y);
        if(err < valor_min)
        {
            obj2.push_back(obj[i]);
            scene2.push_back(scene[i]);
        }
    }
    cout << "Punts depurats primera depuracio obj ->" << obj2.size() <<
std::endl<< "Punts depurats primera depuracio scene ->"<<obj2.size()<<endl;

    Mat H2= findHomography( obj2, scene2, RANSAC, 3.0 );
    std::cout << "Homografia = " << endl << " "  << H << std::endl <<
std::endl;

    FileStorage fh("Homografia2.txt", FileStorage::WRITE);
    write( fh , "Homografia", H2);
    fh.release();

    double h2_11 = H2.at<double>(0,0);
    double h2_12 = H2.at<double>(0,1);
    double h2_13 = H2.at<double>(0,2);
    double h2_21 = H2.at<double>(1,0);
    double h2_22 = H2.at<double>(1,1);
    double h2_23 = H2.at<double>(1,2);
    double h2_31 = H2.at<double>(2,0);
    double h2_32 = H2.at<double>(2,1);
    double h2_33 = H2.at<double>(2,2);

    std::vector<Point2f> obj3;
    std::vector<Point2f> scene3;
    puts("Calcul d'error");
    for (int i=0; i<obj2.size(); i++)
    {
        double err_x = ((scene2[i].x)-
((h2_11*obj2[i].x+h2_12*obj2[i].y+h2_13)/(h2_31*obj2[i].x+h2_32*obj2[i].y+h
2_33)));
        double err_y = ((scene2[i].y)-
((h2_21*obj2[i].x+h2_22*obj2[i].y+h2_23)/(h2_31*obj2[i].x+h2_32*obj2[i].y+h
2_33)));
        double err= (err_x*err_x)+(err_y*err_y);
        if(err < valor_min)
        {
            obj3.push_back(obj2[i]);
            scene3.push_back(scene2[i]);
        }
    }

    cout << "Punts depurats segona depuracio obj ->" << obj3.size() <<
std::endl<< "Punts depurats segona depuracio scene ->"<<obj3.size()<<endl;
    cout << "Impressió d'imatges primera volta de depuració amb matriu
d'homografia" << endl;

    std::vector<KeyPoint> h_kp1, h_kp2;
    std::vector<KeyPoint> h2_kp1, h2_kp2;
```

```
    cv::KeyPoint::convert(obj2,h_kp1);
    cv::KeyPoint::convert(scene2,h_kp2);
    Mat img_h_kp1, img_h_kp2;
    drawKeypoints( img1_r, h_kp1, img_h_kp1, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("img_h_kp1.png", img_h_kp1);
    drawKeypoints( img1_r, h_kp2, img_h_kp2, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("img_h_kp2.png", img_h_kp2);

    cout << "Quantitat de punts:" << endl <<"Keypoints homografia 1 ->" <<
h_kp1.size() <<endl<<"Keypoints homografia 2 ->" << h_kp2.size() << endl;

    Mat h_dt1, h_dt2;
    f2d->compute(img1_r, h_kp1, h_dt1);
    f2d->compute(img2_r, h_kp2, h_dt2);

    puts("Creant matches...");
    Mat img_h_matches;
    std::vector< DMatch > matches_h;
    BFMatcher matcher_h(NORM_L2, true);
    matcher_h.match( h_dt1, h_dt2, matches_h );

    puts("Creant imatges amb matches...");
    drawMatches(img1_r, h_kp1, img2_r, h_kp2, matches_h, img_h_matches,
Scalar::all(-1), Scalar::all(-1), std::vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    imwrite("Matches_homografia.png", img_h_matches);

    cout << "Impressió d'imatges segona volta de depuració amb matriu
d'homografia" << endl;

    cv::KeyPoint::convert(obj3,h2_kp1);
    cv::KeyPoint::convert(scene3,h2_kp2);
    Mat img_h2_kp1, img_h2_kp2;
    drawKeypoints( img1_r, h2_kp1, img_h2_kp1, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("img_h_kp1.png", img_h2_kp1);
    drawKeypoints( img1_r, h2_kp2, img_h2_kp2, Scalar::all(-1),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
    imwrite("img_h_kp2.png", img_h2_kp2);

    cout << "Quantitat de punts:" << endl <<"Keypoints homografia 1 ->" <<
h2_kp1.size() <<endl<<"Keypoints homografia 2 ->" << h2_kp2.size() << endl;

    Mat h2_dt1, h2_dt2;
    f2d->compute(img1_r, h2_kp1, h2_dt1);
    f2d->compute(img2_r, h2_kp2, h2_dt2);

    puts("Creant matches...");
    Mat img_h2_matches;
    std::vector< DMatch > matches_h2;
    BFMatcher matcher_h2(NORM_L2, true);
    matcher_h2.match( h2_dt1, h2_dt2, matches_h2 );

    puts("Creant imatges amb matches...");
```

```
    drawMatches(img1_r, h2_kp1, img2_r, h2_kp2, matches_h2, img_h2_matches,
Scalar::all(-1), Scalar::all(-1), std::vector<char>(),
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    imwrite("Matches_homografia2.png", img_h2_matches);

    waitKey(0);
    system("pause");

    return 0;
}

#endif
```

5.5. Filtratge de punts a partir de la matriu fonamental

```
// fundamental.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include "opencv2/core.hpp"
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <array>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


#ifdef HAVE_OPENCV_XFEATURES2D
#include "opencv2/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"
#include "opencv2/xfeatures2d/nonfree.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2\highgui\highgui.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2\imgproc\imgproc.hpp"
#include "opencv2/core/types.hpp"
#include "opencv2/stitching/detail/autocalib.hpp"
#include "opencv2\core\types.hpp"
#include "opencv2\sfm.hpp"
#include "opencv2\core\matx.hpp"

using namespace cv;
using namespace cv::xfeatures2d;
using namespace std;


int _tmain(int argc, _TCHAR* argv[])
{
    puts(" ///////////////////");
    puts(" COORDENADES TEST 1");
    puts(" ///////////////////");

    std::vector<Point2f>obj_t1;
    std::vector<Point2f>scene_t1;
    std::vector<Point2f>obj2_t1;
    std::vector<Point2f>scene2_t1;

    //Obrir fitxer txt
    FileStorage fq("obj_test1.txt", FileStorage::READ);
    read(fq["obj"], obj_t1);
    fq.release();

    FileStorage fw("scene_test1.txt", FileStorage::READ);
    read(fw["scene"], scene_t1);
    fw.release();
```

```
    //Crear matriu Funamental
    Mat F_t1;
    F_t1 = findFundamentalMat(obj_t1, scene_t1, 3.0, 0.999);
    cout << " Fonamental=" << endl << " " << F_t1 << endl << endl;

    //Guardar matriu Funamental a txt
    FileStorage fe("FundamentalMatrixt_test1.txt", FileStorage::WRITE);
    write(fe, "Fonamental", F_t1);
    fe.release();


    // Això va just després del càlcul de la matriu fonamental
    // vector v1 (u1,v1,1), v2(u2,v2,1) (coordenades imatge (en píxels))
    Mat v1(1, 3, CV_64FC1), v2(3, 1, CV_64FC1);
    // derivades del producte S= vec1t * F * vec2 respecte u1, v1, u2 i v2
    Mat dv1dx(1, 3, CV_64FC1), dv1dy(1, 3, CV_64FC1), dv2dx(3, 1,
CV_64FC1), dv2dy(3, 1, CV_64FC1);

    double res;
    double h = 0.2;
    vector<Point2f>::iterator it2 = obj_t1.begin();
    int i = 0;
    //vector<KeyPoint>::iterator itkeyobj = keypoints_object.begin();

    //vector<KeyPoint>::iterator itkeysce = keypoints_scene.begin();

    cout << " S( u1, v1, u2, v2) = 0" << endl;
    cout << " dS( u1, v1, u2, v2) = 0" << endl;

    double cont_s = 0.0, cont_ds = 0.0;

    for (vector<Point2f>::iterator it = scene_t1.begin(); it !=
scene_t1.end(); it++, it2++, i++/*, itkeyobj++, itkeysce++*/)
    {
        double x1, y1, z1;
        // carreguem v1 i v2 amb les dades del punt vist des de obj i des
de scene
        v1.at<double>(0, 0) = it->x;
        v1.at<double>(0, 1) = it->y;
        v1.at<double>(0, 2) = 1.0;
        v2.at<double>(0, 0) = it2->x;
        v2.at<double>(1, 0) = it2->y;
        v2.at<double>(2, 0) = 1.0;

        // preparem la derivada parcial dS/du1  "les "d" són cursives de
derivada parcial"
        dv1dx.at<double>(0, 0) = it->x + h;
        dv1dx.at<double>(0, 1) = it->y;
        dv1dx.at<double>(0, 2) = 1.0;

        // preparem la derivada parcial dS/dv1  "les "d" són cursives de
derivada parcial"
        dv1dy.at<double>(0, 0) = it->x;
        dv1dy.at<double>(0, 1) = it->y + h;
        dv1dy.at<double>(0, 2) = 1.0;
```